Maël Kimmerlin

Caching in LTE networks using Software-Defined Networking

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 22/09/2014

Thesis supervisor:

Prof. Jukka Manner

Thesis advisor:

D.Sc. (Tech.) Jose Costa

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Maël Kimmerlin

Title: Caching in LTE networks using Software-Defined Networking

Date: 22/09/2014 Language: English Number of pages: 6+68

Department of Communications and Networking

Professorship: Communications Networking Code: S-38

Supervisor: Prof. Jukka Manner

Advisor: D.Sc. (Tech.) Jose Costa

Data consumption is increasing rapidly in mobile networks. The cost of network infrastructure is increasing, which leads to an "end of profit" within next few years. Thus, mobile operators require a new technology that allows increasing the network capacity within low network costs. Therefore, using caching is the most evident solution to be used in their backhaul networks. However, the current architecture of LTE network does not provide sufficient flexibility to place the caches in the most optimal locations. The current work on Software-Defined Networking in Evolved Packet Core virtualization has enabled us to integrate a dynamic caching functionality in a LTE network and improve the caching system performance. In this thesis, we present the solution we designed for this aim based on Software Defined Networking technology. Moreover, we developed a testbed for the proof-of-concept and we present the results and analyze the performances of this solution. We propose three possible optimizations, that we analyze through a simulation.

Keywords: Caching, Software-Defined Networking, LTE networks, Content relocation

Acknowledgements

I would like to express my sincere gratitude to Dr. Jose Costa-Requena for instructing this work. He taught me many things, such as how to proceed in a scientific work and how to write a paper out of my results. This thesis has been a long journey, along which he has always been present to help me. I also want to thank Pr. Jukka Manner for supervising this work and reviewing the thesis. I am grateful to Jesus Llorente Santos and Vicent Ferrer Guasch for their advice, the long discussions on the prototypes and their help to get objective views of my ideas. I also would like to thank my family for financial and moral support.

This work was done in the SIGMONA research project funded by the Finnish Funding Agency for Technology Innovation and Industry TEKES.

Otaniemi, September 2014

Maël Kimmerlin

Contents

Abstract ii

Acknowledgements iii

Contents iv

Abbreviations vi

1 Introduction 1

2 In-Network caching application for mobile networks 5
- 2.1 Performance improvements by caching Http . . . 5
  - 2.1.1 Advantages and drawbacks of web caching . . . 5
  - 2.1.2 Desired properties . . . 5
  - 2.1.3 Caching features . . . 6
  - 2.1.4 Architectures . . . 7
- 2.2 Cacheability analysis for Http in a LTE network . . . 7
- 2.3 Caching location selection . . . 8
- 2.4 Caching optimizations . . . 9
- 2.5 Cache cooperation . . . 10
  - 2.5.1 Virtual Proxy Graph . . . 11
  - 2.5.2 Caching Strategy . . . 11
  - 2.5.3 Adaptation to wireless environment and performance study . . . 11
- 2.6 In-network caching with ICN . . . 12
- 2.7 Summary . . . 13

3 Software-Defined Networking application for caching 14
- 3.1 Applying SDN to mobile networks . . . 14
  - 3.1.1 Application of SDN . . . 14
  - 3.1.2 SDN architecture . . . 16
- 3.2 Using SDN for CCN with a new transport protocol . . . 17
  - 3.2.1 Infrastructure . . . 18
  - 3.2.2 Operations . . . 18
- 3.3 CCN support over a SDN IP network . . . 18
  - 3.3.1 Description of the solution . . . 18
  - 3.3.2 Interaction with usual networks . . . 19
  - 3.3.3 Features . . . 19
- 3.4 An interface between SDN and CCN . . . 20
- 3.5 A content management layer transparent for users . . . 21
  - 3.5.1 Architecture . . . 21
  - 3.5.2 Considerations . . . 22
- 3.6 Summary . . . 24

4 SDN-based caching validation in mobile networks 25
4.1 Solution for the testbed . . . 25
4.2 Caching based on IP addresses . . . 25
4.2.1 Advantages and drawbacks . . . 28
4.2.2 Rules needed on the SDN switches . . . 29
4.2.3 Runtime operations on the SDN controller . . . 30
4.2.4 Caching system requirements . . . 30
4.3 Principle validation . . . 30
4.3.1 Openflow rules . . . 30
4.3.2 Forwarding rules . . . 33
4.3.3 NAT rules . . . 33
4.4 Caching relocation first validation . . . 34
4.5 Issues of the system . . . 37
4.6 Summary . . . 38

5 Caching integration in Software-Defined networks 39
5.1 Forwarding-on content solution . . . 39
5.2 Proxy fully-based solution . . . 43
5.3 Hybrid solution . . . 46
5.4 Simulation setup . . . 50
5.4.1 Description of the model . . . 50
5.4.2 Limitations . . . 52
5.4.3 Results of the first simulation set . . . 53
5.4.4 Results of the second simulation set . . . 57
5.4.5 Forewarn . . . 59
5.5 Miscellaneous optimizations . . . 61
5.6 Charging for the cached content . . . 61
5.7 Summary . . . 61

6 Conclusion 63

References 66

Abbreviations

ACK	Acknowledgement
CCN	Content-Centric Network
CDN	Content Delivery Network
DNS	Domain Name System
enodeB	Evolved Node B
EPC	Evolved Packet Core
ETAG	Entity Tag
FIB	Forwarding Information Table
GGSN	Gateway GPRS Support Node
GPRS	General Packet Radio Service
GTP	GPRS Tunneling Protocol
GW	Gateway
HLC	Home Location Cache
HLS	HTTP Live Streaming
HSS	Home Subscriber Server
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICN	Information-Centric Network
IP	Internet Protocol
ISP	Internet Service Provider
LFU	Least Frequently Used
LRU	Least Recently Used
LTE	Long Term Evolution
MME	Mobility Management Entity
NAT	Network Address Translation
OPEX	Operational expenditure
PDN	Packet Data Network
P-GW	Packet Gateway
PIT	Pending Interest Table
QOE	Quality of Experience
QOS	Quality of Service
RNC	Radio Network Controller
RST	Reset flag (TCP)
RTT	Round-Trip Time
SDN	Software-Defined Networking
SGSN	Serving GPRS Support Node
S-GW	Serving Gateway
SIB	Subscriber Information Base
SYN	Synchronization request flag (TCP)
SYN ACK	Synchronization Acknowledgement flag (TCP)
TCP	Transmission Control Protocol
TOS	Type of Service
TTL	Time To Live
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
URL	Uniform Resource Locator
VEPC	Virtual Evolved Packet Core
VLAN	Virtual Local Area Network
VPG	Virtual Proxy Graph

1 Introduction

Mobile devices such as smart-phones and tablets are more and more widely used by end users. This leads to a fast growth of the data consumption which results in higher bandwidth and resources consumption. The costs are increasing for the operators to create and maintain a network capable of providing internet access to all their end users. Moreover, users are not willing to pay more for their mobile data services. Thus the incomes generated with those equipment are not increasing, which leads to the "end of profit" situation for networks operators in 2015 according to a recent study [1, 2].

Therefore, mobile operators require a solution that will allow increasing the network capacity or optimizing the usage of available resources and reduce the Operative Expenses (OPEX). Caching is already a solution used in wired networks, and widely deployed by Internet Service Providers (ISP) to lower their bandwidth consumption and improve the user experience of their clients. Caches are servers added in the networks, that intercept the requests of the clients in order to serve them from local storage as much as possible. When a request arrives, for a piece of content that is not already stored in the cache, the cache retrieves it from the original web server, stores a copy depending on certain conditions and forwards the response to the client. Any following request for the content will be served with the local copy. By storing the content in a location close to the users, caching permits to reduce the load on the network. The contents are served from a location close to the users, thus the packets travel less, so the network resources are spared. Caching mainly avoids that requests to the same contents go every time through the whole network to the packet gateways and the interconnections. It also permits to reduce the peering costs. From an end user point of view, caching permits to improve the quality of experience (QoE). Since the content is stored closer, the latency is reduced, and the content might be downloaded faster. [2]

The LTE mobile network architecture defines a logical network running on top of a legacy L3 transport network. The latter is providing the connection between the different network elements of the LTE network. The LTE network is usually divided in two parts : the backhaul and the core networks. The backhaul network is the part of the network connecting the access network where the enodeBs (LTE base stations) are located to the core network where the other logical components of the LTE network are located. The main logical components of the core LTE network are :

- The mobility Management Entity (MME) that handles the mobility of the users and controls the handovers between different enodeBs. It handles the tunnel setup between the base stations and the gateways.
- The Home Subscriber Server (HSS) contains all the information needed about the subscribers

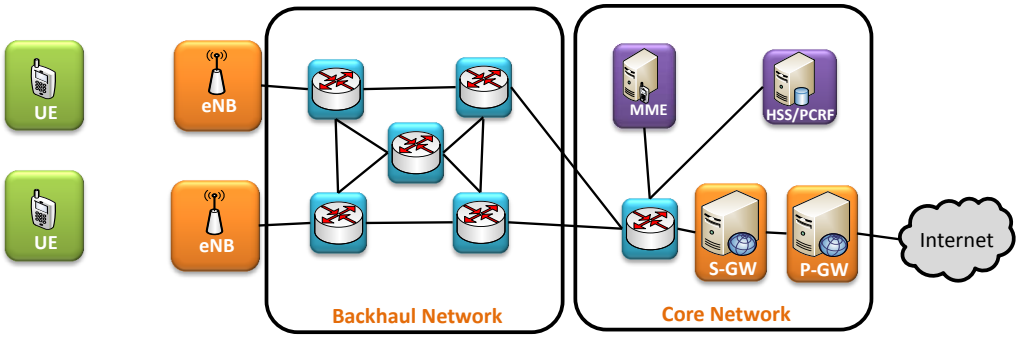

Figure 1: Current LTE network architecture

- The Packet Data Network (PDN) GateWay (P-GW) acts as a gateway to Internet
- The Serving GateWay (S-GW) forwards the data between the enodeBs and the P-GW. The P-GW and S-GW are usually together referred to as S/P-GW.
- There are also other elements such as the Policy Control and Charging Rules Function (PCRF) that applies the QoS and the charging.

Figure 1 shows the current architecture with the access, backhaul and core networks where the LTE logical network elements are deployed. Caching solutions can be applied to LTE mobile network to allow a faster download and spare the power resources of the devices. However, mobile network infrastructure has been designed with tunneling functionality (GTP tunnels) to allow mobility while keeping an active session. Therefore, this tunneling between the base station and the Gateway that connects to public Internet makes it difficult to place the caching in most optimal locations as shown in Figure 2. The caching can be located in both ends of the tunnel which, in practice, means either in the base station or in the core network where the GWs are located. Due to the random and fully distributed content consumption, it is no longer enough to provide an Internet access to the users with centralized caching. Distributed in-network caching could permit even more savings in terms of cost by reducing the bandwidth consumption. Introducing caching in mobile networks also improves the users' experience by reducing the perceived latency. Therefore, the current concern is to move it from this central point to distributed locations in any part of the network that leads to more optimal usage of the network resources. [2]

The current goal is to bring content caching as close to the users as possible but also to dynamically move the cached contents to more optimal location that optimize the overall network bandwidth available. The content would be cached in different places depending on the user demand. Research has been done already on distributed caching in Base Stations for example [3]. Current work on Evolved Packet Core (EPC) in the SIGMONA project [4] permits to propose a new approach to solve this problem of placing the caching in any part of the network. Other works with the same aim could also enable a new approach [5, 6].

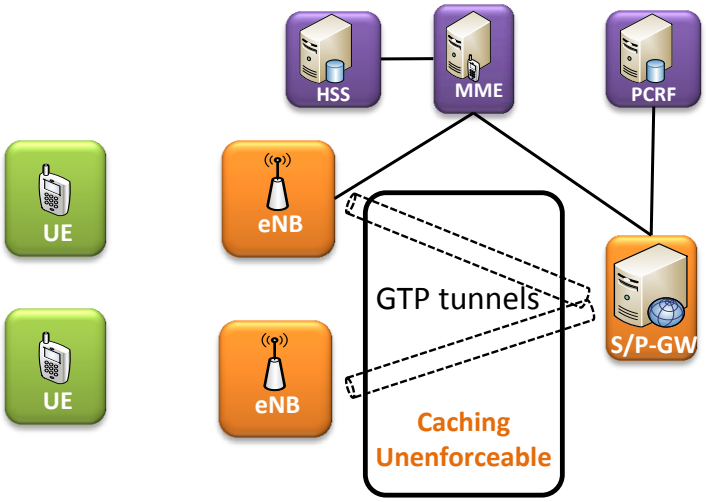

Figure 2: Current LTE Signaling

The work in this thesis explores how an in-network caching system could be implemented in LTE mobile networks. We get started first with the implementation of a testbed of a system working in a legacy IP network. In this implementation, we use software-defined networking to redirect the users' requests to the optimal cache location. This solution uses rules on the Openflow switches to redirect the request to the new cache where the content is located. We use the testbed to measure the efficiency of the proposed design. In our tests, this system decreased the download time of a video from 5.5s when fetching it from its web server to less than 1s when served by the caching system. This system can also reduce the load on the network by nearly 50% for users streaming the same video. Even though the system gives interesting results, it includes some flaws; the main one being the lack of granularity in the control of the content location. There were also some performance issues specific to the implementation. Thus, we propose several optimizations, including different relocation methods. Before deploying the new design in the testbed, we first validate the proposed solutions in a simulation. [2] We found that a hybrid system based on those solutions performs the best and that our time-based relocation system performs better than the solution triggering the relocation decision process based on the arrival of the requests, in terms of load reduction.

We refer to the process that allows fetching the content from the new location of the cache instead of the previous location as a "relocation". With the "relocation" functionality, we are not moving the object itself, which will lead to additional traffic generated to move content between caches, but instead we set the rules in the SDN switches to forward the request to the new location of the content [2].

Chapter 2 presents some previous works related to caching in mobile networks. This previous research investigates the different properties of caching, the possible locations and the features to improve its efficiency. This chapter also presents previous work on virtual Evolved Packet Core (vEPC) that enabled us to propose our solution. In Chapter 3, we explore the possible integration of SDN into caching sys-

tems. The different proposals include several Information-Centric Networks (ICN) or Content-Centric Networks (CCN) architectures. Then we present our prototype, based on those previous works, in Chapter 4. The performance tests results are displayed in this chapter too. Due to the issues of the system, we present some optimizations in Chapter 5. We detail the different possible relocation methods and the setup of the simulation used to analyze them and its results.

2 In-Network caching application for mobile networks

This section presents a diverse panel of possible improvements for caching. We study the design and testing of a distributed system using SDN networking so we do not focus on internal cache content management methods. In this section, similar research performed in this area is described for extensiveness and comparison analysis. In this section, we first present the HTTP caching principles and then some features of the caching system. Finally, we present the optimizations based on which we design our caching infrastructure.

2.1 Performance improvements by caching Http

There are several ways to cache the content, such as peer-to-peer web caching [7] or database caching [8]. The best known is HTTP based caching that is applied in most networks through caches and proxies. J. Wang has published a survey of the http caching schemes [9], where he gives details about techniques to reduce congestion and load on the servers by caching the web content.

2.1.1 Advantages and drawbacks of web caching

The first and most relevant advantage is that caching reduces the network traffic and the bandwidth consumption. The requests are served from a local copy, thus they do not go across the whole network to fetch the content from Internet. Thus, caching reduces the network load and the peering costs since less volume is going in and out of the network. Another advantage is that serving the content from local copies improves the users' experience by reducing the latency. The cache allows retrieving the content from a close location so the round-trip time (RTT) is lower for the local cache than for the original web server.The caching also increase reliability since the connection does not go across multiple links through the uncertain Internet. Another positive point is that caching reduces the load on the servers by load-balancing the requests on the different caches. Thus caching can also enhance the robustness of the service since in case the web server goes down, the contents can still be served with the local copy stored in the cache. However, caching can have drawbacks too, since it increases the latency in case of a cache-miss (content not stored yet) or in case of non-cacheable content. Caches can also be a bottleneck or a single point of failure if the design is not done carefully. Finally, the caches modify the content retrieval information.

2.1.2 Desired properties

The implementation of the caches should comply the following recommendations:

- Fast access : to ensure a low latency.
- Robustness : not to be a single point of failure.

- Transparency : for the users not to care about it.
- Scalability : to support the load.
- Efficiency
- Adaptiveness : to adapt to the demand changes
- Stability : to avoid disruption in services.
- Load balancing : for scalability and stability reasons.
- Simplicity : for the system administration.

These items are fundamental to implement an efficient caching system.

2.1.3 Caching features

The cache systems can be highly complex and this section provides an overview of the different features that can be implemented in a cache. The first feature when implementing a caching system is the resolution procedure. Resolving caches follows a similar process to DNS resolution, that maps a content identifier such as an URL to a caching location in case of distributed caching. There are two techniques to locate the content. The first one is a cache routing table containing pointers to the location of the content. This table is updated using multicast communications between switches. The second option to locate the cached content consists of a hashing function that gives the location based on the URL and the cache membership list. In both cases, updates are needed if the caches network changes.

Moreover, there are also two solutions such as prefetching and cache placement that can be used to improve the cache-hit ratio. The prefetching is done by retrieving future popular files based on a prediction algorithm. The prefetching can be performed at different levels, either between user and proxies, between proxies and web servers or between users and web servers. The second solution consists of cache placement or replacement. It can use traditional policies such as Least Recently Used (LRU) or Least Frequently Used (LFU). The LRU removes the file that was not accessed for the longest time. The LFU removes the file that is the least frequently accessed. Modified versions of those algorithms that consider the properties of the files, such as their size where the biggest files are removed or LRU with a minimum size are removed, are new approaches to solve the problem. Another algorithm can be based on the latency for retrieving the file so the files with the lowest retrieving latency are removed.

Removing files leads to the cache coherence problem. Caches need a way to remove outdated pages. There are two ways to ensure cache consistency. One way is by using revalidation, such as conditional GET including ETags, or server invalidation. Conditional GET is a kind of Http request similar to the GET request but including a "If-modified-since" header. This header contains the timestamp of the version currently cached by the client. If the resource on the server has been modified since this time, the server sends the new version, otherwise the server sends

a "304 Not Modified" header. This method is quite expensive because it implies to contact the server every time the resource is downloaded by the client. The other revalidation way is called server invalidation. It is a process where servers update their clients, but this means keeping track of them, thus increasing the load on the server itself. Those solutions are not simple and instead weak solutions such as a time-to-live for the documents may be preferred. The TTL is based on the time since the content has been sent and not outdated. The TTL defines a time during which the cached version can be served directly. But this static solution may lead to problems in case of high rate of changes for the resources. Another solution is to require a validation of the already stored content of a web server when accessing it, so that all the contents from the same server are revalidated or discarded at once.

2.1.4 Architectures

The different features described above are used to increase the efficiency of the caching system. Another possibility for improving the caching system efficiency is to distribute the system so that caches can cooperate. There are two main and a third hybrid architecture achieving this objective. One of the main architecture is the hierarchical where caching takes place at every level of the network and a request will go through the caches from the access level to the gateways level until it finds a cached copy, otherwise the request will be forwarded to the web server. The reply is cached in all the proxies the request crossed. In this solution, coordination is needed between the different levels and multiple copies are hold at different levels. Moreover, overload may happen on high level caches, with a queue that may increase too much the latency.

The second main architecture is the distributed that is only composed of low level caches. These caches keep metadata about the content of each other. Even if the requests that are cache-misses are retrieved from the other distributed caches, metadata stored by the caches are still hierarchically distributed. The advantage of this solution is that the traffic remains in the access network. However, if the size of the distributed system is too big, connection times may be unacceptable. Te third hybrid architecture with a limitation on the cache cooperation can ensure that the time needed to serve the content remains smaller than the one needed to fetch the content from the web server.

The remainder of the thesis focuses on the distributed architecture type. The research is focusing on caching in the mobile networks since it can reduce the load, mainly in the backhaul network, and reduce the latency for the users. Solutions such as proactive caching and distributed caching have been proposed. This section presents these techniques, focusing first on the cached content and the location of the caches, and then on the caching techniques.

2.2 Cacheability analysis for Http in a LTE network

The performances of the caching systems are tightly related to the type of content targeted for caching. B.A. Ramanan et al. have published an analysis of the content

cacheability in a LTE network [10]. Their study defines what content could be cached in the Base Stations to reduce the bandwidth consumption. To be cacheable, an object must fulfill requirements on its http header fields:

- Set-cookie must be null.
- Vary must be null.
- Content-Length must not be equal to 0.
- Last-Modified must not be set to 0.
- Cache-Control must not be "private" or "no-cache".

Moreover, only the responses with the codes 200, 203, 206, 300, 301 and 410 can be cached.

With these criteria, 31% of the requests in their samples were cacheable, and 74% of the http volume was cacheable. The type of content such as images, video, audio, text or application, also impacts the cacheability. The images are easy to cache, their rate of revisit is high and this permits to reduce the latency and the bandwidth consumption. Moreover, few revalidations are needed. The text contents have a high rate of revisit too, but a low data volume. It reduces the latency but does not change the bandwidth consumption. Their revalidation rate is quite high. The application contents are similar, but they have a higher revalidation rate. A piece of content with a high revalidation rate should not be cached since the cache will probably send a request to the web server to revalidate the content. Caching such content can only spare bandwidth. Thus, audio contents are not a good target for caching because of their very high revalidation rate. Video contents are a good target, since there are few requests and a huge data volume so caching videos spares bandwidth but does not reduce latency. For saving bandwidth, videos should be cached, while images, text and application contents caching would reduce latency.

Moreover, the time an object is cached impacts the performances. Thus, caching images a longer time than other contents can improve the system performances since their maximum age is usually over 24h. For the other resources, it is less possible to do so since the maximum age is much lower. Even if we focus on these cacheable contents, we ensure that the performances for non-cacheable contents are not degraded.

2.3 Caching location selection

This thesis focuses on the caching location, that has economic impacts; for example, in the network of a Universal Mobile Telecommunications System (UMTS) [11]. The base stations are not the best place for caching content with the usual caching techniques. In a mobile network, each caching place such as GGSN, SGSN, RNC or base station, has both advantages and drawbacks. Caching in GGSN or SGSN is centralized, so it is easier to manage but it does not reduce the load on the backhaul network. On the contrary, caching in RNC is decentralized, so it spares bandwidth

and reduces latency, but it is more difficult to manage and there are less cache-hits since there are less users accessing each cache. Caching in the base station is very close to the end user and may both spare bandwidth and reduce latency, but the cache-hit rate is quite low since there are few users.

Cache-hit ratio and traffic costs including the transport cost and the devices cost are two criteria for the comparison of these locations. The best savings are done in SGSN but base station caching performs the worst savings.

However, this study only focuses on usual caching locations and does not include distributed caching methods. Some techniques can improve caching in general. For example, decentralized caching improves caching when getting closer to the users.

2.4 Caching optimizations

S. Woo et al. used samples of a mobile internet provider in Seoul to compare caching strategies such as usual web caching, prefix-based caching and TCP redundancy eliminating caching [12]. From the traffic provided, 95% is using TCP protocol and 75% is using HTTP. The usual web caching can save up to 1/4 of the downlink bandwidth with an infinite cache but the load reduction decreases quickly with the cache size. The prefix-based caching technique removes duplication based on the hash of the first N bytes of the cached content and the content length, but does not improve the performances more than 5% and induces serious problems with false positive. Depending on the value of N, it increases false positive or reduces process speed. Finally, the TCP Redundancy Elimination solution includes two middle-boxes A and B. B divides the incoming TCP flows into small chunks of data and sends it to A that stores the chunks, reconstitutes the flow and sends it to the user. In this solution, the TCP flow caching can be managed easily. This solution permits to save up to 40% of the bandwidth.

Improvements of the caching performances can be done either on the contents themselves or on different aspects such as cache relocation, that can also improve decentralized caching.

K. Y. Lai et al. proposed a cache relocation solution [13]. The Home Location Cache (HLC) is a per-device cache on base stations but this system has some limitations linked to the user's mobility. For example, when changing cells, if the cache does not move with the user, it may be too far from the client and cannot be used any longer or might introduce higher latency and delays. Path prediction could be used to predict the next position of the user and move the cache to the new base station, but it may be sent to the wrong cell if the prediction is not accurate enough.

Two-phase relocation A two-phase relocation mechanism has been designed to fix these flaws. When a user changes the base station, the cache content is copied to a parent node of the current cache and a defined percentage of the content of the current cache is also copied to the predicted future base station. Thus, if the prediction is wrong, the cache content is in the parent node, closer to any new base station and is retrieved from there. If the prediction is right, the rest of the content is retrieved. The content is then suppressed from the other places.

The choice of the parent node takes into account the cost of the move and the probabilities for the new cell so that it minimizes the costs in both case of wrong and accurate predictions. The percentage is chosen with regard to the minimum hit ratio which must be achieved for the client. This relocation method has interesting properties considering the relocation process, but is tightly linked to the devices themselves. This method is optimized to move contents following specific users and does not offer a global view on the requested contents, leading to high duplication level in the HLC.

Return-path Object List Passing In the two-phase relocation solution, the contents must be moved from one cache to the other while there might be some copies stored in other locations that are closer to the destination base station. Hence, the Return-path Object List Passing (ROLP) solution consists in sending a list of the objects the cache contains, from the cache of the original base station to the caches that are in the possible next base stations. Thus, the new cache can retrieve the content from the closest location, and not from the previous cache. This solution permits to save bandwidth and permits some cooperation between the caches. These solutions are improving the performances of the HLC but do not fit in our context.

Pre-loaded caching alternative Å. Arvidsson et al. proposed a solution based on pre-loaded caching to improve decentralized caching [14]. The pre-loaded caching solution supports both pooling and equalizing. Pooling consists in retrieving content in one cache from other caches and equalizing in pushing objects from one cache to the other to preload contents. A central controller manages content by keeping track of the requests and of the contents location. The controller can be reactive or proactive. If the controller is reactive, in case of a cache-miss, it defines the location from which the content must be retrieved. If the controller is proactive, it predicts which content will be popular in the close future and defines which objects should be pre-loaded in the caches during the off-peak hours, based on estimations of the requests.

Pooling always increases cache-hit ratio, and full equalization improves the performances. When using equalization, cache-hit ratios are similar in all the cells. As a side effect, equalization may lead to saturation in extreme cases due to load on the network that might be unnecessary if the predictive algorithm is wrong or performed during a high-load period. Moreover, this system needs a transient time to fill the caches, during which they do not perform as well as usual caches and generate big amount of traffic. Pre-loading caches is an efficient method of improving the performances of the system. Thus, we decided to include in our optimized design a similar feature so that, when using a predictive algorithm for the relocation, the system could pre-load the contents that will be requested in the optimal location.

2.5 Cache cooperation

Usual caching techniques includes hierarchical cooperative proxy caching or vertical cooperative caching. However, these caching schemes are not efficient in mobile

networks since these techniques cannot handle the handover of the clients and do not focus on the costs between proxies. Thus, J. Z. Wang et al. have proposed a cooperative caching model for Base Stations [15]. The proposal is similar to a previous work by J. Z. Wang and V. Bhulawala [16].To fix the issues of hierarchical or vertical caching systems, they propose a peer-to-peer cooperative proxy caching scheme. This scheme is built over a new cache line containing a cached document and references to the cache that previously stored the object in its header.

2.5.1 Virtual Proxy Graph

In this case, the caching scheme is based on a Virtual Proxy Graph (VPG), to ease the data linkage and transfer. The VPG contains all the nodes and the links between them. Thus, each proxy can determine which nodes are its neighbors and communicate exclusively with them. A centralized control is not needed and the whole proxy network is automatically adjusted since proxies use broadcast messages to update their characteristics. The broadcast messages permit to fill in the neighbor table, from which the caches select the neighbors they will communicate with to create a virtual link. This selection is based on the distance between each other and the impact on the user, to keep the latency reasonable. Each proxy sets up its copy of the VPG.

Thus, the whole proxy graph is known by all the proxies and may be self-adjusted with other broadcast messages. For example, if the communication is too low on a virtual link or too high without a virtual link, proxies may agree to drop or create the virtual link and will send a broadcast update of the VPG.

2.5.2 Caching Strategy

In proxies, the new cache line can be in three states for each piece of content:

- The full cache line is stored; the document is served when requested.
- Only the header is stored; then the cache will retrieve and store the document from the closest cache indicated in the header if it takes a shorter time than fetching it directly from the web server.
- Nothing is stored; the cache will send a broadcast request. If any cache has the line or its header, it will send the header. Every proxy that forward the reply will also update its own table of headers. Then the proxy will process the request as in the previous case. If not, the line is created and the content fetched from the web.

The replication in this design is on-demand only, the caches are storing only the requested content while performing a high load reduction.

2.5.3 Adaptation to wireless environment and performance study

To best fit the wireless environment, there is a need for geographical information during the self-configuration. Such information includes for instance the physical

distance between the base stations. However, the self-configuration must be mitigated by self-adjustment, for example, if two Base Stations are close but no user move from one to the other, the proxies will drop the virtual link. Compared with caching strategies such as cache relocation, multicast-based cooperative caching or peer-to-peer cooperative caching, the peer-to-peer cooperative caching with cache line migrations performs the best in terms of cache-hits and latency since the headers are smaller than the full documents. The cache line migration consists in moving the headers when the user changes of base station. But this caching scheme does not perform better considering the volume of data exchanged in comparison with the same caching scheme without cache-line migration since there are more movements. However, the increase of cache hits justifies the load increase.

This caching scheme is fully decentralized. Due to the current centralized nature of SDN protocols such as OpenFlow, we have been considering a distributed caching system with a centralized cache controller where we could apply those techniques. The equivalent of the VPG and the cache lines are stored in the cache controller. We implement pooling and later equalization as optimizations to our caching system.

Such distributed caching, with or without a centralized controller, is applied in new types of networks such Information Centric Networks (ICN) or Content-Centric Network (CCN), where it has a prominent role.

2.6 In-network caching with ICN

ICNs are applying caching to a wider range of protocols, not only to HTTP. They gather new approaches of networking that focus on content, including Content-Centric Networking. Instead of being based on a point-to-point communication concept, CCN is on the contents rather than on their positions. There are already several proposals for implementing this new approach [17]:

- The "clean slate" proposal is aimed at implementing CCN directly over the layer 2 networks, thus replacing the IP protocol.
- The "overlay" proposal is aimed at implementing CCN over IP networks using TCP or UDP.
- The mixed approach proposes an extension of IPv4 and IPv6 to include ICNs.

The overlay proposal is close to the aim we want to achieve with our system, thus in Chapter 3 we detail the different solutions to implement ICN on top of SDN. The remainder of this section presents the Information Centric Networks in more details.

In any of those three proposals, the packets should be routed based on the content name and not on its location. This routing-by-name is composed of two parts [17]: the forwarding-by-name, transferring the packet to its output interface based on the content name in each network equipment and the content routing, spreading the content location information over the network.

Caching becomes a central part of ICNs. Caches bring the content closer to the users transparently, using content retrieval based on the contents names and no

longer on the contents location, independently of any application specific caching system [17]. G. Rossini and D. Rossi proposed a performance study for caching in CCN. CCN is defined as a model with users, proxies and caches. The users request contents with "interest packets". If the contents are stored, they are sent directly. Otherwise, the name and the interface on which the request came are added to the Pending Interest Table (PIT). The request is forwarded based on the Forwarding Information Table (FIB). Thus, if a node receives several requests for a same piece of content simultaneously, the requests are aggregated if the content has not yet been received. The corresponding PIT entries are removed when the reply is received. Content is also cached when going through the nodes [18].

The forwarding parameters impact the performances of this type of networks. In CCN, routing and forwarding are decoupled, routing being considered as modifying the FIB and forwarding using the FIB to send the packets. The behavior of the nodes when there are several caches containing the data chunk is an influential parameter. The possible solutions tested are to forward the request to the closest cache, or to forward it to all the caches. Forwarding to the closest cache reduces the possibility of aggregation for the responses, that is a key point of CCN, and forwarding to all the caches induces more competition between caches and thus a faster eviction for content.

The problems ICNs try to solve also include an adaptation to the biased distribution of the requests, by some factors such as the language of the users. In Finland for example, if there is a region speaking mostly Swedish and a region speaking mostly Finnish, the contents fetched will not be the same, e.g. the news websites will be adapted to the language of the users.

All those solutions implemented in ICN or CCN are targeted at improving the performances of caching systems. We believe that including Software-Defined Networking in this architecture will also participate in this aim.

2.7 Summary

Caching is defined as storing a local copy of contents from the web and serving the requests from the copy instead of the original . It is an efficient way to spare bandwidth and network resources, as well as improving the user experience by reducing the latency and increasing the speed of the downloads. There are several methods of improving the efficiency of caching systems, most of them are based on a distribution of the service. For instance, cooperation between caches increases the cache-hits and the overall performances of the system. Hierarchical cooperation has been implemented as a first cache cooperation system, and currently more efficient proposals include neighbors in different ways to increase the number of cache-hits. Finally, caching has been placed as a central feature for a new generation of networks, the Information-Centric Networks. There have been several attempts to base such systems on SDN in order to keep improving the caching systems. Hence, the next section focuses on SDN and its applications in caching and in LTE networks.

3 Software-Defined Networking application for caching

Research has focused on Software-Defined Networking as a component of caching systems, for example Software-Defined Networking in Information-Centric Networking. This chapter presents the different solutions proposed. First, it describes the principles of SDN and a direct application in mobile networks. Then we present how SDN is used in the context of the SIGMONA project to remove the GTP tunnels and enable caching in the backhaul networks. The remainder of the section presents different ways to apply SDN in ICN or CCN.

Software-Defined Networking implements a separation between the control plane and the forwarding plane of a network. Figure 3 shows the dissociation between the application layer, the control plane and the forwarding plane (hardware switches). This architecture makes the control plane of the network entirely programmable. Moreover, the control of the network is centralized. Thus, the control plane is dissociated from the switching equipment since it is managed by the centralized controller [4]. All the decisions made at the controller level will be applied on the forwarding plane (switches). Thus, changes do not have to be done on each piece of the network, but once in the controller that will apply it to the underlying infrastructure. The control layer is thus independent of the hardware components and provides the network services and API for applications that would not be available in usual network components. Openflow is one implementation of this architecture [19].

With the current focus on content-centric networks, researchers on SDN have been focusing on how to introduce a software-defined networking solution, such as Openflow, in mobile networks.

3.1 Applying SDN to mobile networks

CellSDN was designed by L. E. Li et al. as an infrastructure using Software-Defined Networking in mobile networks [6]. The goal of CellSDN is to simplify the design and the management of cellular networks by using SDN features. In a legacy LTE network, the Serving Gateways (SGW) handle the changes in the user location and tunnel the traffic to the Packet network Gateway (PGW) that applies the fine-grained policies such as firewalls, Quality of Service and billing, defined by the Policy Control and Charging Function (PCRF). Base Stations (BS) can also use the subscriber's information to reduce the rates when it gets congested. Communication between them and the Mobility Management Entity (MME) is getting more and more complicated, for handover for example. So the complexity is increasing in a network that is not evolving any longer. SDN could solve the problem.

3.1.1 Application of SDN

In order to use SDN in mobile networks, some adjustments to SDN are required:

- The policies based on the subscribers' information must be translated into packet-matching rules.

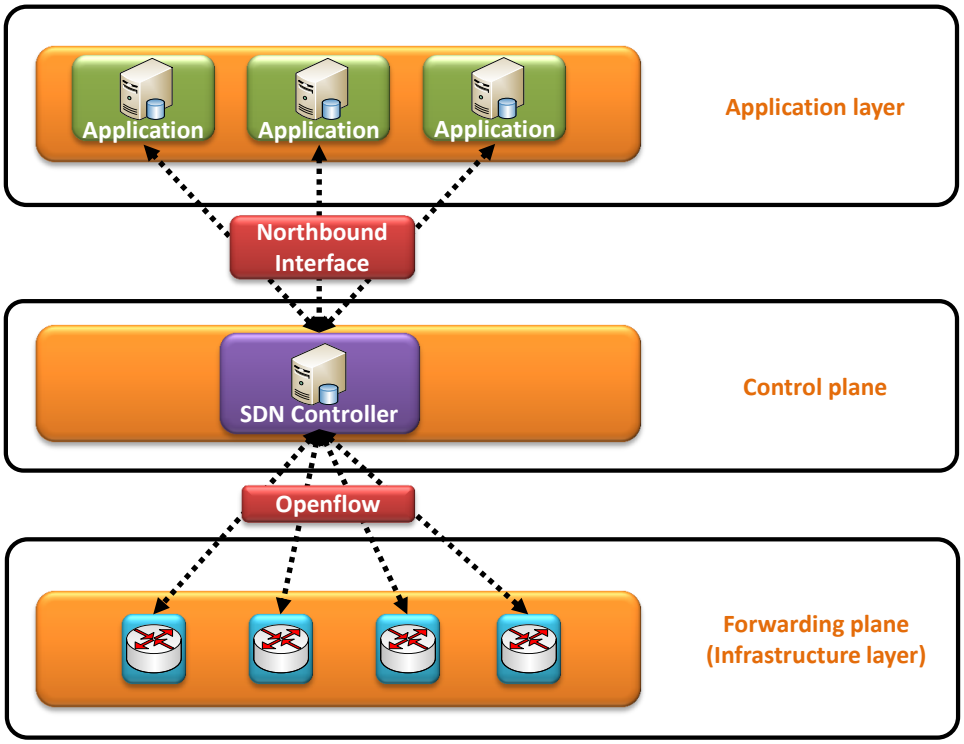

Figure 3: Software-Defined Networking Architecture

- Some mechanisms of the SDN controller should be delegated to reduce the load of the controller, for example, counter pulling can be done by local agents.
- Some actions should be added to the switches such as header compression/decompression.
- The network virtualization should be based on the subscribers' information.

With such adaptations, SDN can have several applications in mobile networks:

- Directing the right part of the traffic to middle-boxes such as Intrusion Protection System or echo suppression for VoIP, and if the switch has some DPI extensions, it could even provide the functions by itself.
- Monitoring for control and charging. For example, a SDN controller with real-time monitoring can adapt the network to the changes such as global users' mobility and thus load-balance the traffic.
- Providing a seamless mobility for the user. SDN can control the routing over different technologies that would otherwise require long delays and complex mechanisms when the user switches of technology.
- Providing QOS and control access policies. That is currently done in PGW but could be distributed over all the switches with SDN, thus enforcing the policies also for the traffic remaining in the cellular networks.

- Providing network virtualization. LTE does not permit to fully virtualize the networks but SDN could provide isolation for each carrier for example, or for roaming customers and usual subscribers. Each part of the network could then be managed independently.

- Providing a centralized and decoupled control of the base stations with a hardware API.

3.1.2 SDN architecture

These features are proposed as extensions of the current SDN implementations. In order to translate the policies based on the subscribers' information, the first extension includes several modules to get the users' location, the paths between the different network equipment, etc. The modules need the subscriber's information to adapt to the users for roaming, rates, echo cancellation if needed. The controller maps the subscribers with user IP addresses by using a Subscriber Information Base (SIB). Finally, the SDN controller uses this table to translate policies based on the user to policies based on packets.

The second extension is aimed at offloading the controller. Local agents on the switches can handle events that may be difficult for the controller to handle, or would use too many resources, such as adaptation to congested networks. The agents can also perform some predictable actions such as pulling counters. These agents have a limited control on their environment so that they can solve some of the problems independently, and faster than the controller.

The third extension focuses on the patterns and actions of the switches. Larger rule tables are needed for fine-grained policies and DPI modules. They also include features such as header compression and decompression.

The last extension is aimed at virtualizing the networks based on the subscribers' attributes. Thus, it is possible to slice the network resources, and base station resources as slicing the semantic space, to isolate roaming subscribers for example.

However, the main application of the SDN controller in this work has been proposed in the SIGMONA project. Caching in the backhaul network can only be performed on top of a legacy IP network. In current backhaul networks, the traffic is encapsulated in GTP tunnels. SDN permits to remove these GTP tunnels that were preventing the operators from caching in backhaul networks. Previous work in the SIGMONA project has focused on integrating SDN in LTE network with this aim. In this design, the S/P- GW is removed and a SDN based packet network is used instead with the SDN controller interacting directly with the MME to set up the rules on the SDN switches. The MME handles the handovers by notifying the SDN controller to change the rules on the SDN switches.

Double vlan encapsulation (802.1ad) is used instead of GTP tunneling. The first vlan tag is used as a service tag to identify the operators and the second is used as a customer tag to set up tunnels between the enodeB and the IP routers at the edge of the network. This tunneling can be handled by SDN switches [4].

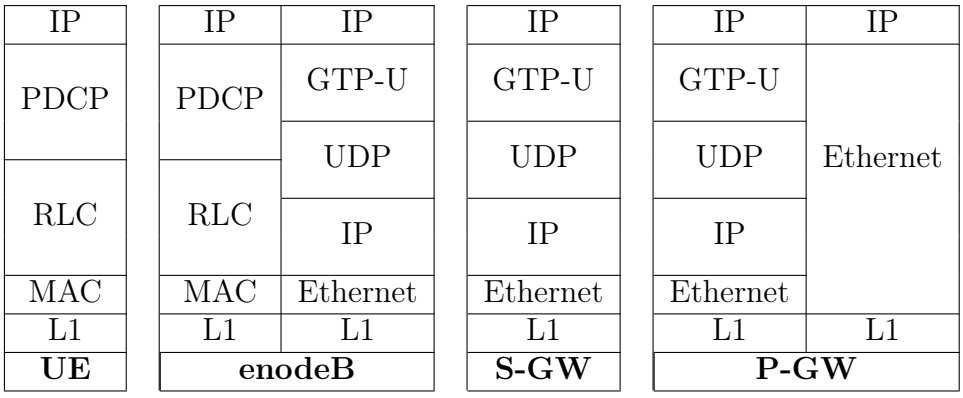

Figure 4: Current LTE stack [4]

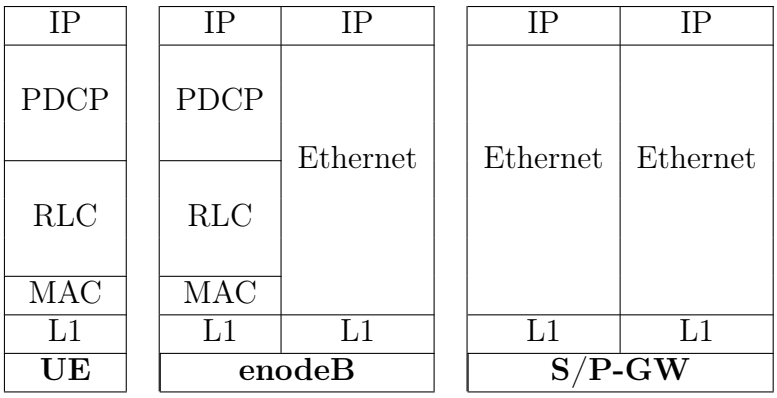

Figure 5: New LTE stack [4]

Table 4 presents the current protocol stack in LTE networks and Table 5 presents the new protocol stack when using SDN. It clearly shows that all the encapsulation and the tunneling layers have been removed and that the Layer 3 is indeed the client IP layer. Thus we can work with the clients IP packets in SDN switches. [4]

SDN can also be used to build caching systems. Current research has been focusing on using SDN in Information-Centric Networks or Content-Centric Networks to maximize the performances.

3.2 Using SDN for CCN with a new transport protocol

A first solution to join SDN and CCN was proposed by N. Blefari-Melazzi et al. Their design is based a new IP transport protocol and is not interoperable with legacy networks. Gateways are used to connect with other networks on which only TCP or UDP are supported [17].

3.2.1 Infrastructure

The infrastructure contains different types of nodes: end nodes that correspond to the users, border nodes that are taking in charge the forwarding of requests and responses, based on names and internal nodes that are caching nodes.

The client will first issue a request and send it to the closest border node. The border node will forward it based on the name of the content, and will fill its Pending Interest Table (PIT) with the name of the content and the port the request came from. The request will reach other border nodes or internal nodes. There, it will either be a cache-hit or a cache-miss. If the object is stored in the internal node, then the content will be sent. If the object is not stored, the request will be forwarded by the rules installed by the controller. In this infrastructure, every node, even the border nodes, can have a cache for content.

3.2.2 Operations

A Name Resolution System (NRS) maps the content name with the redefined port fields of the packet so the content can be easily identified. The responses of the NRS are cached in the nodes. A single request to the NRS for a piece of content is needed until the timeout of the response expires. Then the ports combination may be reused.

The ingress border node will send the incoming user's request to the controller that will query the NRS to get the identifier of the object. Then the controller will set the rules to forward either to the correct cache or to the egress switch if the content is not cached. All the requests and responses are sent within IP packets for transfer between nodes.

Due to the mapping between content and ports, only 2^{32} pieces of content can be requested at once. Moreover, this solution uses a custom transport protocol. So other solutions have been proposed to solve these issues.

3.3 CCN support over a SDN IP network

M. Vahlenkamp et al. proposed an infrastructure supporting of the ICN/CCN features such as routing based on names and aggregation. The main constraints of this solution are to respect the existing protocols and to allow interconnection with usual networks (not SDN). The controller of the network has an overall knowledge of the content of the caches [20].

3.3.1 Description of the solution

All the ICN transfers are made over the IP protocol, using either TCP or UDP. Figure 6 presents the Operation flow of this design. When receiving an ICN request, the first switch will identify it based on a special port or transport protocol and will send it to the controller. In the network where this solution is applied, IP headers are not modified but they are used differently. The source IP field contains the message identifier, the source port contains the reference of the switch on which the request

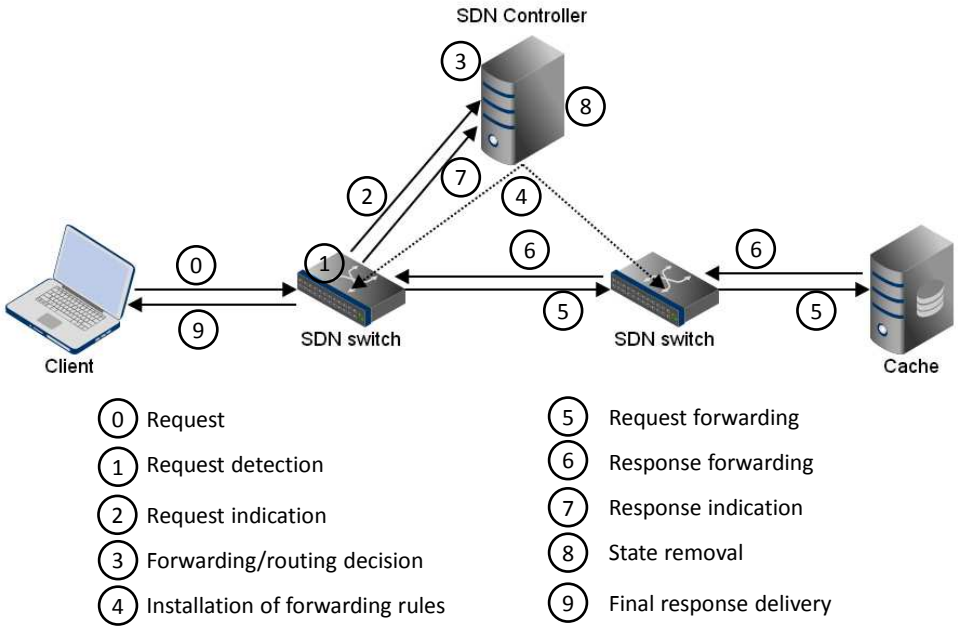

Figure 6: ICN Operation process [20]

first arrived and the destination IP field contains the IP of the cache containing the requested content. The destination port contains a reference identifying the packet as part of an ICN request or response. Thus, if the packet is coming from or going outside of the network, a header rewriting operation is needed at the ingress or egress switch. The controller performs the mapping needed for the header rewriting. When receiving a packet from outside the network transmitted by the ingress switch, the controller will store the source IP and port and will generate a message identifier. It will then change the IP header accordingly and send it back to the ingress switch while installing the rules for the flows to forward the packets to the cache containing the requested content. The controller needs an exhaustive knowledge of the contents of the caches. If a piece of content is not found, the request may be forwarded to the original web server.

3.3.2 Interaction with usual networks

This infrastructure can interact with networks that are not SDN networks due to the header rewriting. The whole network may be identified outside by a single IP address and a special destination port identifying ICN requests or a source port for ICN responses. The request or response will then be routed using this IP address as destination or source in the packets on the usual networks. The header rewrite operations permit this interconnection.

3.3.3 Features

Typical ICN features such as request or response aggregation are enabled. The controller can define some forking switches where the content is duplicated in case

of several identical requests. The SDN controller can perform these operations because of its extensive knowledge of the network and the content of the caches. Thus it can drop duplicated requests and fork the responses to satisfy the users while reducing the network load. Moreover, if the content is not found, the request can be immediately forwarded, depending on the network policy. The balance between network load and controller load can be adapted through system configuration.

3.4 An interface between SDN and CCN

Wrappers are another solution to provide CCN features over SDN. An example of such an interface between usual SDN switches and CCN nodes has been designed by X.-N. Nguyen et al. The wrapper provides an abstraction layer used between Openflow and CCNx, in order to enable name-forwarding [21, 22].

The wrapper is an intermediate layer that interacts with both CCNx and Openflow without any modification to the existing software. The aim of the wrapper is to encapsulate content in an IP packet where the header fields such as IP addresses or ports are based on the hash of the name of the content. Thus, Openflow can perform forwarding based on the name. This solution uses hardware capacities natively, for Openflow for example. However, with this mapping on the header fields, the original fields lose their meaning. This is not a flaw if the whole network is consistent, applying this solution, otherwise header rewriting is needed to interconnect with other legacy networks, as in the previous solution.

Each CCNx node is connected via a wrapper to a SDN node. When a user sends a request or a server send a piece of content, the Openflow switch will set the TOS (type of service) field of the packet accordingly to the port the packet came in and will send it to the wrapper. If it is a request for content, called interest for CCNx, the wrapper will use the TOS to send it to the right face of the CCNx instance. A face is a virtual port corresponding to the host port on the switch. If it is some data, the wrapper will send it directly to the CCNx instance on a dedicated face. The reply of the CCNx instance can be either pieces of data or another request in case of unsatisfied user request. If the reply is a piece of data, it will be sent on the face corresponding to the destination host. Then the wrapper will accordingly set the TOS based on the incoming face and the IPs based on the content name and will send it to the switch. If the reply is a request coming from the dedicated face, the wrapper will set the TOS to 0, and send it to the switch. The switch will then send it to the right port according to the TOS. If the TOS is 0, the packet will be forwarded to a configured default interface on the switch.

This design decreases the performances, around 5%, but enables a centralized control of the Pending Interest Tables and Content Store. However, by using original CCNx nodes, this solution is tied to the CCNx project so currently, the users have to use dedicated software.

3.5 A content management layer transparent for users

ContentFlow is a transparent solution designed by A. Chanda and C. Westphal. ContentFlow is based on the HTTP protocol [23] and aimed at providing a content management layer, corresponding to the Openflow controller, that manages contents, maps each of them with a location and takes in charge the routing or forwarding based on the name of the contents. Their solution is aimed at being transparent for the users and for other networks, thus interconnection is possible without gateways.

3.5.1 Architecture

ContentFlow is composed of proxies, caches and a controller. Since the solution is using http, proxies are needed to establish the TCP connection. Figure 7 shows the architecture and process this solution.

When sending a request, a client will first try to connect (TCP) to the web server. It is mandatory to use proxies to establish the connection with, instead of the real server, because the content request is sent once the connection is established. So when a user is willing to send a request, his first TCP packet (SYN flag) that do not match any flow on the switch will be sent to the controller. The controller will create some static rules to redirect the traffic of this flow to the proxy. Then the connection will be established and the user will send its request. The proxy will analyze the request and create a handle composed of the file identifier, the source and destination IP addresses and ports. The proxy will send the handle to the controller to query the location. Then the controller will look up where the content is stored, if any.

If the content is not cached, the controller will determine whether it should be cached and where. Then the handle will be sent to the selected cache and the controller will create a forking point for the reply on the switch that is the best located considering the proxy and the cache, to minimize the load on the network for the forking operation. It will install the static rules for the flows and reply. After receiving the reply, the proxy will forward the request to the actual web server. The reply is then duplicated at the forking point; a copy is stored in the cache and the response is sent to the user who requested it.

If the file is found in the caches, the controller will send the IP of the cache to the proxy that will forward the request to this cache and the reply to the user. In order to perform such a process, the proxy is adapted. It must be able to communicate with the controller. This communication is performed through a REST API.

The cache must be adapted too since it does not receive both requests and responses but only the responses. So it must receive the information metadata from the controller. It also has to use the Content-Length header in case of partial content e.g using the HTTP response code 206 to know when the whole response has been received. For these metadata, the controller has two dictionaries, one mapping the file names and the cache IP addresses and the other one mapping the content names with the server IP addresses and ports. The mapping with the file names is used to find if the content is cached and where. The mapping with the server information is compulsory to set up the rules for the flows.

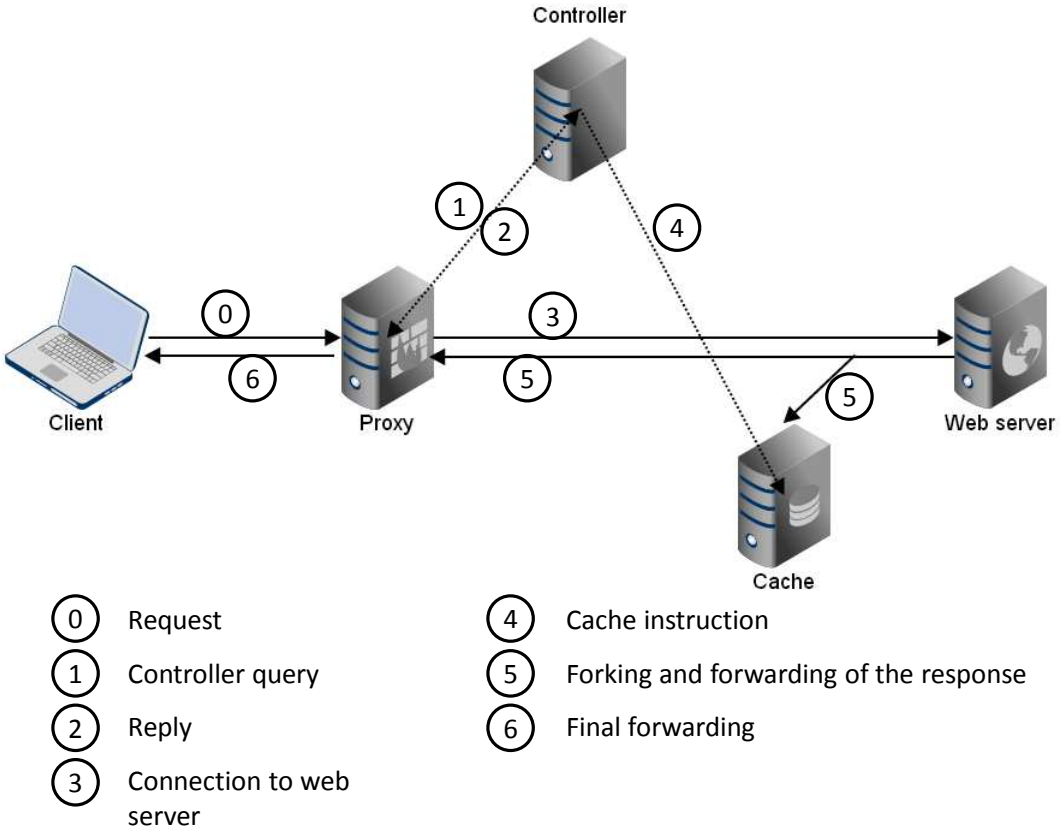

Figure 7: ContentFlow process [23]

3.5.2 Considerations

The file naming is the native scheme of HTTP which can ensure the uniqueness of a name and a standard method for parsing. However, there is a necessary mapping between the file name and some fields of the IP and TCP headers. The size of the fields in the TCP and IP headers is limited, so the number of possible mappings is limited too. Hence, the fields are freed in order to be reused later. The caching policy applied can be adapted to take some parameters into account, such as popularity.

Y. Sakurauchi et al. had previously proposed a similar solution, redirecting requests to proxies depending on either the IP or the URL [24]. In the case of URL-based redirection, since http is using TCP, the controller let the client do the TCP handshake with the server then the http request is redirected to the proxy and forwarded to the cache. The problem then is that the connection is not established with the cache, thus a RST packet is sent to the client. So there are two connections needed and this decreases the performances of the system. This flaw was fixed in the ContentFlow solution by also redirecting the TCP handshake to the proxy as in the IP-based redirection. Figure 8 presents the IP-based redirection. Every packet with a new IP address is sent to the controller that selects a cache location and instructs the switch to forward the packets to the proxy. Every following packet is then forwarded to the same location. However, most of the websites have several IP

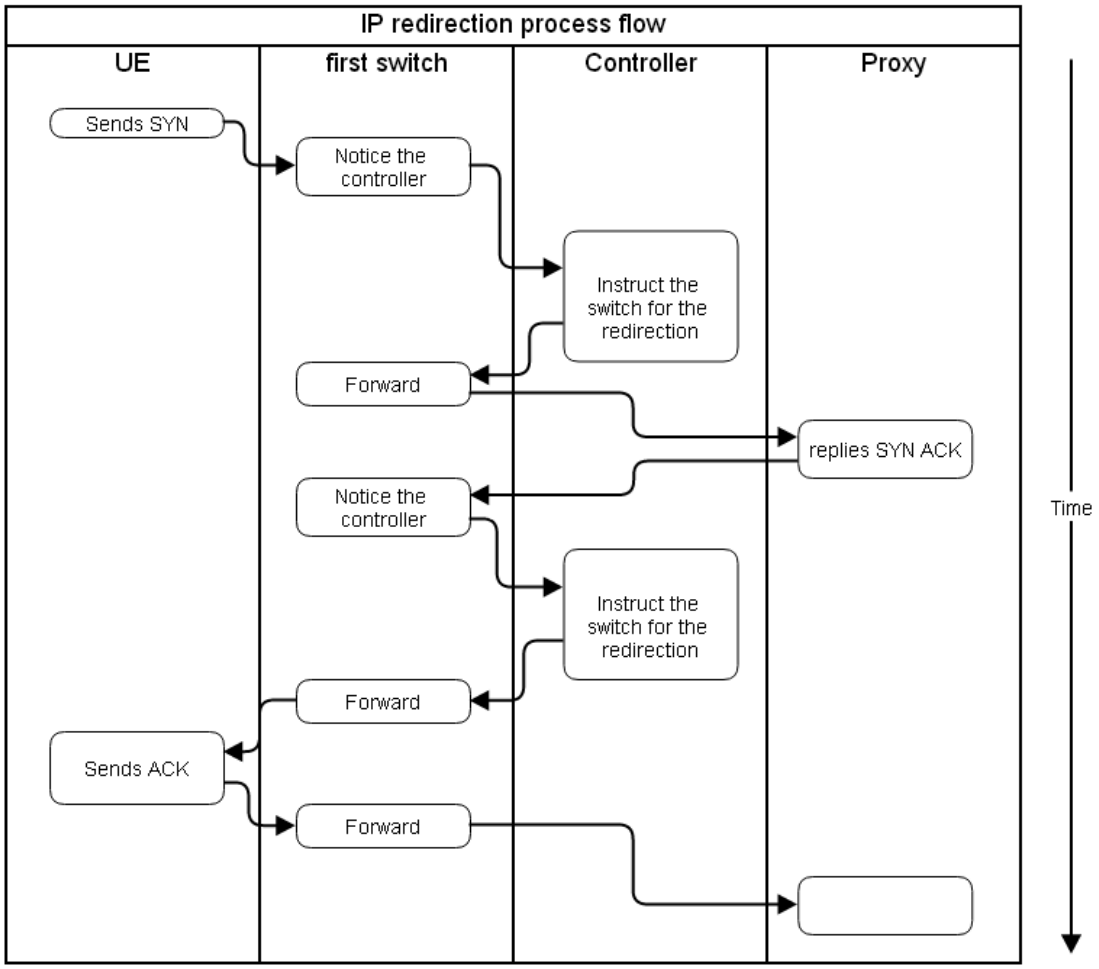

Figure 8: IP-based redirection [24]

addresses but in the controller, when receiving the notice for the SYN packet, it is not possible to relate it to any previous request, even if it is for the same content with a different IP. Thus there is no control over what content is stored in the caches, thus where the contents are stored and over the contents replication.

Considering the different solutions, we have selected the contentFlow solution. It has the main advantage to use the TCP/IP protocols without modifying them, only modifying the routing or the forwarding of the packets with the SDN switches. In our work we decided not to implement an elaborated solution such as a wrapper since our concern is only about HTTP. This protocol is already natively supported over the TCP/IP networks. To build our first implementation for validating the concept in mobile networks, we consider the work of Y. Sakurauchi et al. about redirection based on IP addresses.

3.6 Summary

SDN can have multiple applications either in LTE networks or in ICN / CCN. In LTE networks, SDN can be used to increase the performances of the networks with applications such as redirection or seamless mobility. In the SIGMONA project, SDN is used to create a virtual EPC. This vEPC permits to remove the GTP tunneling and thus to set up caches in the backhaul networks. We are aiming our project at adding values to bringing SDN in LTE network. Thus we aim to use SDN to improve the caching in LTE networks. SDN has already been used to improve performances in ICN or CCN networks, either in systems built using new protocols or working over a TCP/IP network. There are several proposals but due to our constraints such as using a legacy IP network, we selected one of the solutions as a basis to build our new system. In the next section, we will present the new architecture and describe the validation tests and their results.

4 SDN-based caching validation in mobile networks

This chapter presents the implementation of the prototype and the tests we performed to validate the concept. In this section, we first present the environment of the tests, the implementation and the results of these validation tests.

4.1 Solution for the testbed

The architecture used for the testbed has been inspired by the ContentFlow solution. It is composed of several elements, including caches, proxies, a SDN controller and a cache controller. The proxies include a request analyzer, to get the URL requested, and is thus referred to either as a proxy or an analyzer. In this testbed, the topology is defined and does not change so some related static rules are set on the switches. They are detailed in Section 20.

Figure 9 presents the first testbed, composed of a single client, a proxy, a SDN controller, a cache controller and two caches. The tests were performed with three configurations:

- Without any caching, directly fetching the content from internet.
- With caching in a first cache.
- With caching in a cache closer to the client.

The tests have been performed by adding latency and reducing bandwidth on the link between the two parts of the network [25].

4.2 Caching based on IP addresses

This proposition is our first attempt to use SDN in a caching system designed for LTE backhaul networks. Limiting the number of intermediary elements such as proxies when processing the requests was a requirement. In this proposition, the caching is done on a per IP basis. We uses the destination IP address of the packets to redirect them to the cache in charge of this website.

Figure 10 shows the process of our solution. The first request of a client to a website that was not already reached (1) is redirected to the proxy (2). The proxy gets the host by analyzing the request and sends it to the cache controller (3). The cache controller will associate all the IP addresses it gets by resolving the hostname to a cache location. Then the cache controller set rules in the first switch after the enodeB to redirect to the cache in a specific vlan all the HTTP traffic with those IP addresses as the destination (4). The vlan tag will be removed at the last switch (4). Then the proxy can either forward the request to the cache by himself (6) if the controller inserted the rules needed, or send a response with a redirection to the client, that will then issue a second request to the same IP. Even if the second solution is not optimal in terms of users' experience, we implemented it for the sake of simplicity in our proof-of-concept testbed. This flaw has been corrected in the optimizations proposed. Our design is based on network address

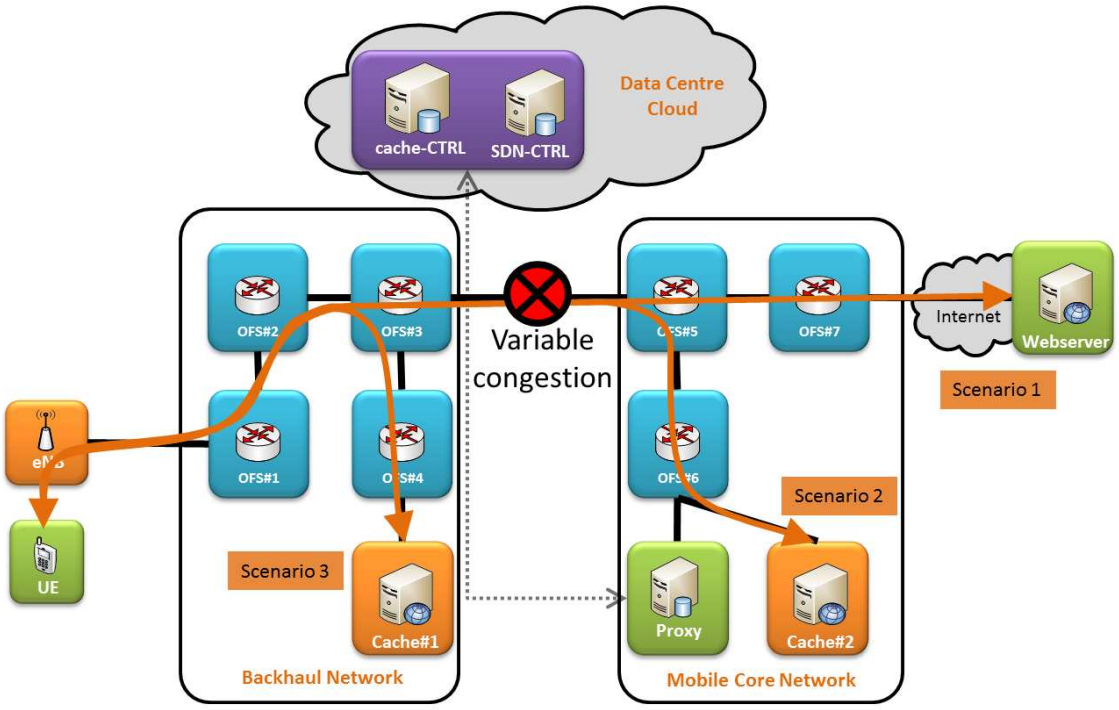

Figure 9: Demonstration scenarios of the SDN caching implementation

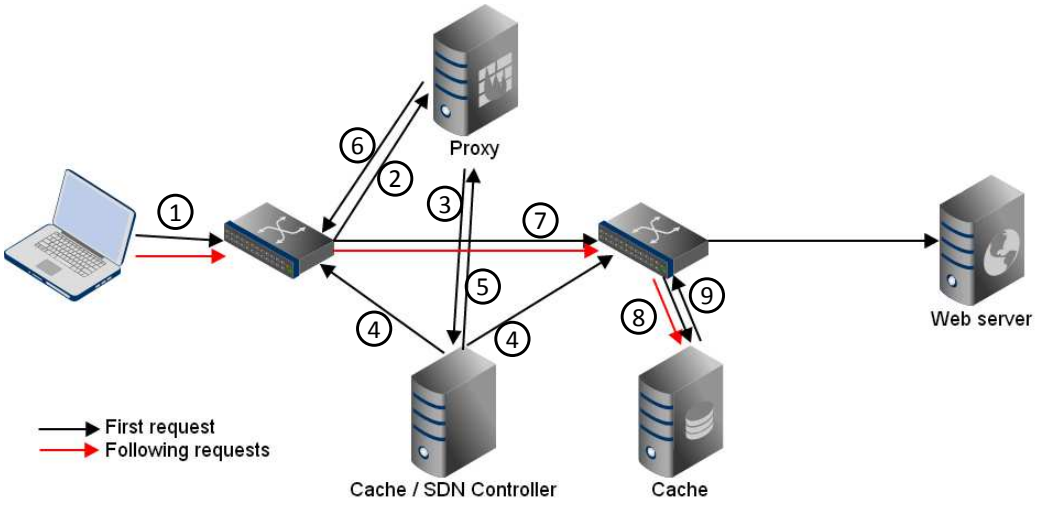

Figure 10: Components and interaction of the caching system

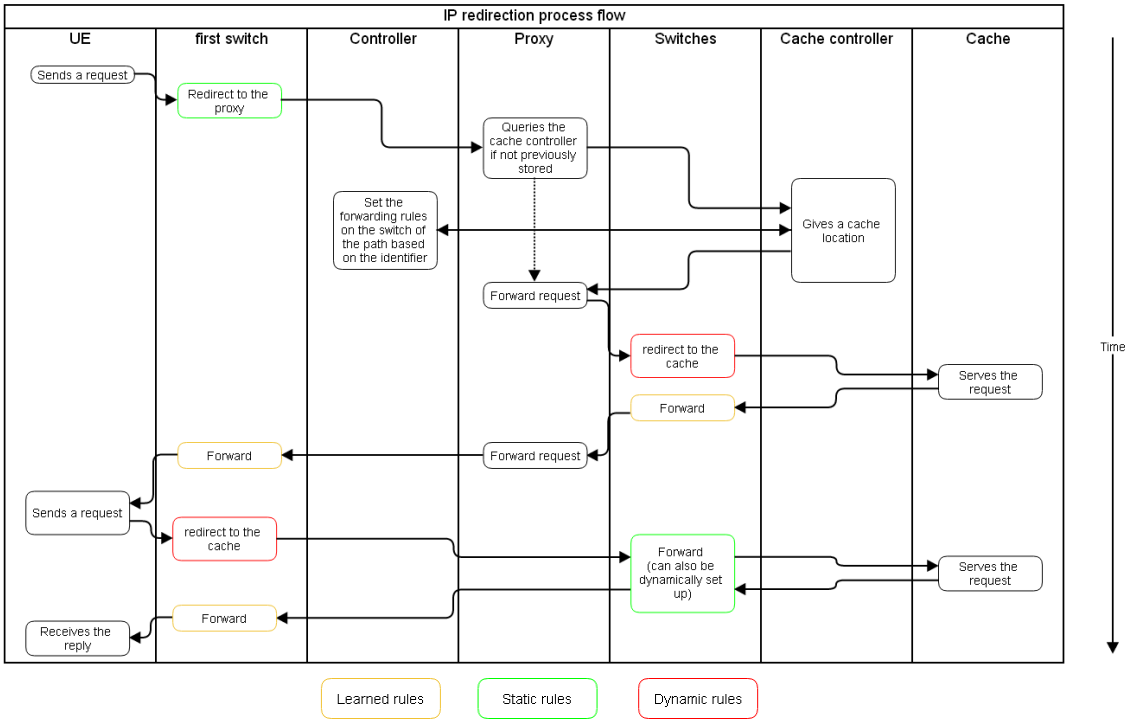

Figure 11: process of the UE request

translation (NAT) on the destination address, to redirect the packets to the right cache (7)(8), in order to be interoperable with legacy IP networks and routing or forwarding processes. Changing the routing or forwarding process, instead of using NAT, is another possible solution. This method requires to add higher priority rules matching the IP addresses and ports that should be redirected to output the packets through a specific port. This would imply a higher number of operations, with a higher number of switches to configure, and requires a fully SDN-compliant network. Moreover, the load due to the number of rules would be on all the switches, not only the switches at the base stations. Thus, we selected the NAT based solution. [2]

Currently, building a webpage in a browser includes requests to 17 different domains on average [26], for purposes such as advertisements or social modules. Those domains can also be associated to several IP addresses with different servers using DNS round-robin. Thus, a webpage request can lead to a high number of operations, but once the setup done, any following request to the same elements will not lead to any further operation.

Figure 11 presents the mobile's request process and Figure 12 shows the relocation process for this solution. Currently, the relocations are triggered manually. In further sections, the relocations are triggered based on time. We define a time interval between two sets of relocations. The impact of this period will be described in Section 4.4.

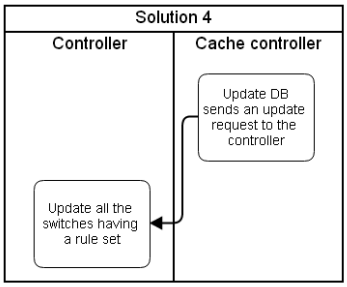

Figure 12: relocation process

4.2.1 Advantages and drawbacks

We designed our prototype to ensure the following advantages [2]:

- A single SDN switch is needed per enodeB . If the use of the vlan is not usual, such as in the SIGMONA vEPC, another SDN switch can be required at the connection points of the caches.
- A reduced number of configuration operations is required, one per IP address on the switch of the enodeB and some static rules set up beforehand.
- Since most requests are made to the same websites, e.g. Facebook, Google: Gmail, Youtube etc., Yahoo, Wikipedia etc. and thus to the same IP addresses, most rules can be set for long times. Only the websites that are not often accessed need new rules. The frequency of new rules setup is rather small when the rules for the most accessed websites have been set up.
- Relocating content implies only a single operation per switch at the base stations.
- The system does not need big computation power. Since it is not fine-grained, the rules are set per website, including thousands of contents. So the number of needed operations is divided by the average number of contents per website.

But the drawbacks are quite important [2]:

- The contents of a website are not separated. Due to the redirection mechanism, the whole website contents have the same caching location. It is not possible to apply fine-grained policies to favor some Youtube videos in their location among others, for example.
- Relocation can only be performed for the whole website, moving all the contents stored from the same website at once. It is not possible to relocate only the most accessed content. Thus the relocation operations take a very long time and are not efficient.
- Contents of the cache are not controlled on a fine-grained level.

- Duplication is not controlled. If duplication is required, it is performed on a website basis, all the contents are duplicated.

The inaccurate handling of contents is the main problem of this solution and will be further detailed in Section 4.5.

4.2.2 Rules needed on the SDN switches

This solution uses mainly two kinds of rules:

- A first static rule is needed per base station to redirect the http traffic to the proxy. Depending on whether the proxy is transparent or not, the switch might learn new rules from this one. These learned rules are dynamic.
- One rule per IP of the accessed websites, per switch where is connected an enodeB on which users accessed the content, is used to set the vlan and the destination Mac and IP if needed to perform the NAT.

Table 13 presents the rules for this solution.

Switch	Match	Action	Aim
First after the UE	TCP, port 80	Set dst IP and Mac to proxy, output, learn [match (TCP, dst IP = src IP, dst port = src port, input port = output), action set src IP, set src Mac]	Redirect to the proxy, learn a rule to set back the original IP based on the src IP and port
First after the UE or last before the cache	Dst IP, TCP, port X	Set dst IP and Mac to cache, output, learn [match (TCP, dst IP = src IP, dst port = src port, input port = output), action set src IP, set src Mac]	Forward to the cache, learn a rule to set back the original src IP and Mac address of the reply
First after the cache or last before the UE	Dst Mac, dst IP and dst port, input port	Set src IP, Mac and port	Set the webserver IP as src IP (learned rule)
All	Dst Mac, Dst IP, dst port	Output	Forwarding rule (optional)

Figure 13: rules on the SDN switches

4.2.3 Runtime operations on the SDN controller

The number of operations performed at runtime varies depending on the method used in the implementation. If the network is using NAT on the destination address to redirect the packet, the runtime operations consist in setting the redirection rules to perform the nating and learn the reverse rule, i.e. one operation per IP address for new websites accessed per enodeB. If the network uses higher priority rules to route the packets to the cache, more rules will be needed, at least one per switch crossed per IP address. Moreover, the number of relocation operations will also vary to modify all the rules concerning a website either on every first switch at the enodeB affected or on all the switches where a rule was set to route the packets with the particular IP address.

4.2.4 Caching system requirements

- Full knowledge about the caches and base stations network is required.
- Knowledge of the cache content is not mandatory, but could benefit the system by optimizing the storage possibilities.

4.3 Principle validation

4.3.1 Openflow rules

Rules on the switches are divided into two categories, static or dynamic rules. Both are set up by the controller. The static rules are set up when the switch connects to the controller. They usually do not have any timeout. The dynamic rules are set up at runtime, either for a limited time, or until another operation from the controller. We will discuss the rules set up on the switches and therefore, some precisions are needed beforehand.

Some rules are set to provide a similar service as Network Address Translation (NAT) on destination IP addresses. The aim of the rules is to change the Mac and IP addresses as needed, to the addresses of the target of the redirection operation. This action can be taken with a single rule, but the reverse operation is also needed in order not to break the IP layer compatibility. To be able to set back the previous destination MAC and IP addresses, the switch should have memorized it. Moreover, the switch can only match on the IP and MAC addresses and the destination port that is the client port. The origin port is the server port and is 80 for all the packets of http transactions in this system. Thus, the outgoing NAT rule includes a learn action that will create a new reverse rule for each new connection going through, so that the expected IP and MAC addresses can be set back on the incoming packets. Those rules will be short-lived but there might be as many such rules as there are clients' connections redirected through this process, and the rules not deleted yet but not used any more since the connection was closed, but the timeout was not reached yet. It can go up to several thousands of rules. So the static redirection

rules imply a high load on the switches where they are applied. The learned rules presented in this work are limited by the user's browsers themselves. For example, the number of TCP connection in Firefox is limited to 256 by default [27], so if all the users are using Firefox with its default configuration, there will be an upper bound of 256 rules per user per learning action.

Figure 14 presents the rules organization in OpenFlow switches. In our prototype, the static rules are used to set the default vlan, if needed, for the content coming on a port and strip the vlan tag, if needed, before sending it on the tagged port. There is also a default static rule with low priority for all the base station ports, that redirects any non-matching packet with TCP port 80 to the default proxy associated to the switch.

The controller is setting the dynamic rules for the forwarding and the redirection. For the forwarding, the whole system behaves as a network of simple layer 2 learning switches. The rules on the switches are divided into different tables, depending on their role in the packet processing. One of the tables is used for the forwarding process. If a packet does not match any rule in this table, meaning that the destination or origin Mac address is unknown, the packet is sent to the controller. The SDN controller adds the addresses to its database with the correspondence to the ports and adds two rules on the switch in both ways, to process the packets between those two hosts.

For the redirection, the TCP packets with destination port 80 are redirected by default to a proxy. The proxy will parse them, get the desired URL and the host IP and then send a request to the cache controller to set up a rule for redirecting the packets for this host to a cache. The cache controller will query the SDN controller to set up the needed rules. The cache controller will provide the IP and Mac address of the selected cache. The SDN controller will finally set the rules to modify the destination Mac and IP addresses with the one of the caches and reversely for the reply packets. The reverse operation is based on the port number and IP address of the client to identify to which IP the client is theoretically to be connected. The SDN controller sets learning rules on the switches [2]. These rules create new rules based on the association between the web server IP and the port of the client for every matching packet. These new rules must be short-lived or overwritten because they change every time the TCP connection is closed and a new connection is established with the same source port.

The proxy parses the incoming http request. Currently, it resolves the name of the desired resources and adds rules through the cache controller for every address returned, if the request is susceptible to have a cacheable answer. Then it closes the connection with a redirection HTTP code. The proxy could instead forward the request to the original web server and its reply to the client.

The cache is a server that first analyzes the incoming request and either replies with a cached object or fetch it from the original web server and add it to its storage. The library used for caching in this implementation is CacheControl [28]. It is a complementary project to the module Requests [29]. Our first implementation is in Python and has severe performance issues.

In the next section, the rules are organized by tables, depending on their role,

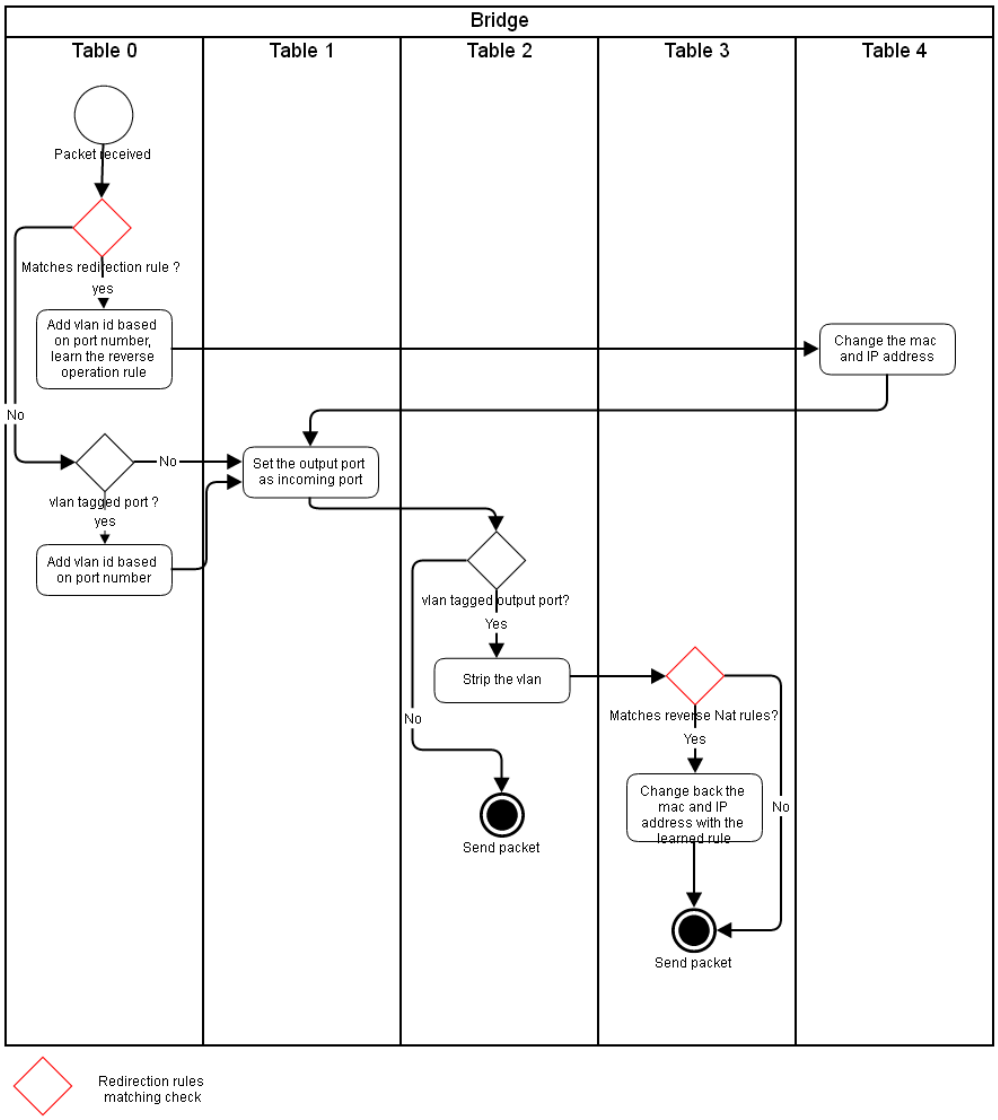

Figure 14: Rules organisation in OpenFlow switches

for the forwarding or for the modification of the Mac and IP addresses.

4.3.2 Forwarding rules

Table 0 Table 0 is used to set the vlan tag if needed to the incoming packets, depending on the incoming port. The first rule sets the vlan tag and the second rule does not change the packet:

```
#cookie=0x0, table=0, priority=1, in_port=1 actions=mod_vlan_vid
    :1,resubmit(,1)
#cookie=0x0, table=0, priority=1, in_port=2 actions=resubmit(,1)
```

Table 1 Table 1 is used for the forwarding rules. It is basically a L2 forwarding with two rules per pair of hosts. The rules are matching the Mac address.

```
#cookie=0x0, table=1, priority=1, in_port=2, dl_src
    =00:00:00:00:00:06, dl_dst=00:00:00:00:00:03 actions=resubmit
    (1,2)
#cookie=0x0, table=1, priority=1, in_port=1, dl_src
    =00:00:00:00:00:03, dl_dst=00:00:00:00:00:06 actions=resubmit
    (2,2)
```

If a packet matches one of those rules, it is resubmitted to the next table with its destination port as incoming port

Table 2 Table 2 is used to strip the vlan if needed:

```
#cookie=0x0, table=2, priority=1,in_port=1 actions=strip_vlan,
    resubmit(,3)
#cookie=0x0, table=2, priority=1,in_port=2 actions=resubmit(,3)
```

Table 3 Table 3 is used for outputting the packets or for performing other operations by the NAT rules.

```
#cookie=0x0, table=3, priority=1,in_port=1 actions=output:1
```

4.3.3 NAT rules

Table 0 For the NAT process, the packet matches a rule of higher priority. The table 0 in the NAT process is used to learn the new rules for the reverse operations. Then the packets are resubmitted to table 4 that takes in charge the NAT process, before resubmitting them to Table 1.

```
#cookie=0x0, table=0, priority=2,tcp,in_port=1,tp_dst=80 actions=
    mod_vlan_vid:3, learn(table=3, priority=2, NXM_OF_ETH_DST[]=
    NXM_OF_ETH_SRC[], eth_type=0x800, NXM_OF_IP_DST[]=
    NXM_OF_IP_SRC[], nw_proto=6, NXM_OF_TCP_DST[]=NXM_OF_TCP_SRC
    [], load:NXM_OF_ETH_DST[]->NXM_OF_ETH_SRC[], load:
    NXM_OF_IP_DST[]->NXM_OF_IP_SRC[], output:NXM_OF_IN_PORT[]),
    resubmit(,4)
```

This rule will create a new rule in Table 3 that will set the destination IP address of the packet as the source IP address of the packet that will match this new rule. The mac addresses will be changed too. The new rule will match on the client IP address and port. The rule presented below is the default one to redirect to the proxy. The rules to redirect to the cache are similar, with a higher priority.

```
#cookie=0x0, table=0, priority=2,tcp,in_port=1,nw_dst=10.0.0.1,
   tp_dst=80 actions=mod_vlan_vid:6,learn(table=3, priority=3,
   NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[], eth_type=0x800,
   NXM_OF_IP_DST[]=NXM_OF_IP_SRC[], nw_proto=6, NXM_OF_TCP_DST
   []=NXM_OF_TCP_SRC[], load:NXM_OF_ETH_DST[]->NXM_OF_ETH_SRC[],
    load:NXM_OF_IP_DST[]->NXM_OF_IP_SRC[], output:NXM_OF_IN_PORT
   []), resubmit(,4)
```

Table 4 The table 4 performs the NAT operations and resubmits the packet to table 1 for usual forwarding:

```
#cookie=0x0, table=4, priority=2, tcp, in_port=1, nw_dst
   =10.0.0.1, tp_dst=80 actions=mod_dl_dst:00:00:00:00:00:04,
   mod_nw_dst:10.0.0.4, resubmit(,1)
```

The only difference between the default rules for the redirection to proxy and the rules for redirection to the caches is the priority, except the Mac, IP and ports, so that the rules for the caches are matching first, then the rules for the proxy, and then the usual rules.

Table 3 Tables 1 and 2 are not impacted by the NAT process. In case of nating, rules are added in table 3 for the reverse operation by the learn action of rules in table 0:

```
#cookie=0x0,  table=3, priority=2, tcp, dl_dst=00:00:00:00:00:03,
    nw_dst=10.0.0.3, tp_dst=56508 actions=load:0xaad94633d5ab->
   NXM_OF_ETH_SRC[], load:0xa000001->NXM_OF_IP_SRC[], output:1
```

The original IP and Mac addresses are set back and the packet is sent on the output port.

4.4 Caching relocation first validation

We first performed simple tests in order to check the validity of our concept. We set up a client in the base station and downloaded a video of 2.4 MB coming from the Internet, with caching successively in cache 1, in cache 2 and no caching. Then we added latency at the congestion point to simulate congestion. The results are presented in Figure 15.

Without caching, the download time varies because of variations in the Internet and is overall quite high. Without congestion, the two caching locations behave similarly in terms of download time. When increasing the congestion, the further cache performs worst than the closer one, up to performing like the direct download,

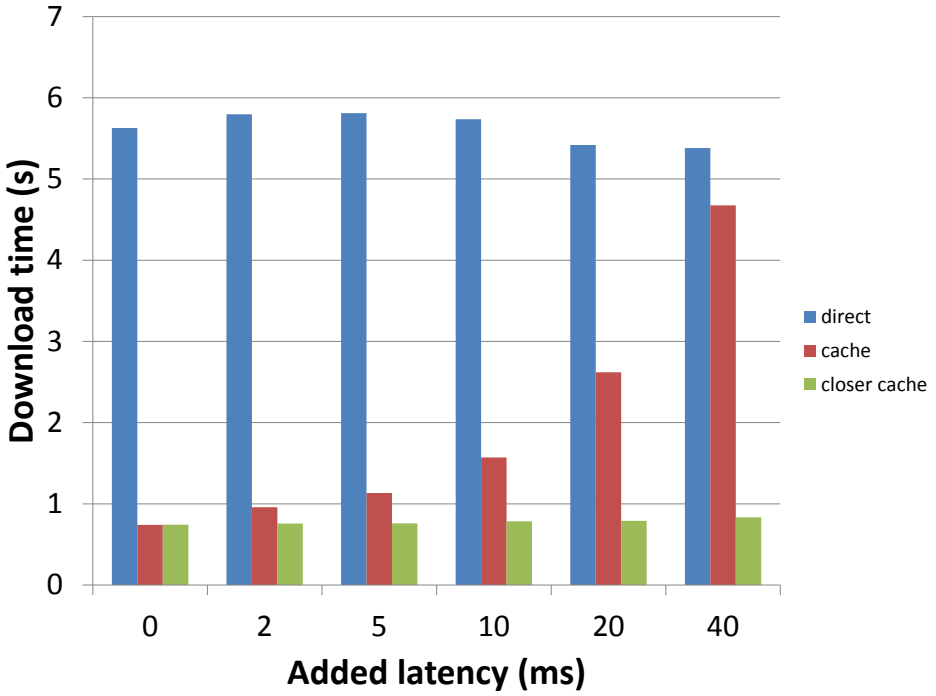

Figure 15: Download time depending on the latency and caching location [25].

while the closer cache remains fast for the download. Figure 15 shows well the aim we want to achieve with our system.

The first tests enabled us to validate the idea of caching relocation using SDN with a simple Python implementation of the proposed design. The next tests were performed with contents from the internet and videos from a local server, using Http Live Stream (HLS) to download the video chunk by chunk from an Http server.

HTTP Live Stream (HLS) is a streaming protocol created by Apple. It allows clients to download videos over HTTP/HTTPS, with all the features of HTTP/HTTPS such as authentication and encryption. Thus, a simple web server is needed to offer the video, without any extension or plugin needed. The stream is chunked using a segmenter and can be proposed with different bitrates. An index file lists all the chunks available. The clients can download each chunk as needed, and adapt to the available bandwidth [30].

In order to prove the efficiency of the proposed design, we set up a second testbed, Figure 16 shows this testbed. We performed different test cases using a HLS video that was downloaded by the mobile clients. The test cases include several users distributed between different base stations. The clients are streaming this video with a 2Mb/s bitrate. We consider two base stations with seven users, 5 on the first and 2 on the second. The SDN switches are OpenVSwitch [19] bridges and the caches are still implemented with CacheControl. We run different test cases which consist of adding artificial congestion, i.e. we use netem and qdisc commands to increase the delay in the selected openvswitch links, in some segments of the

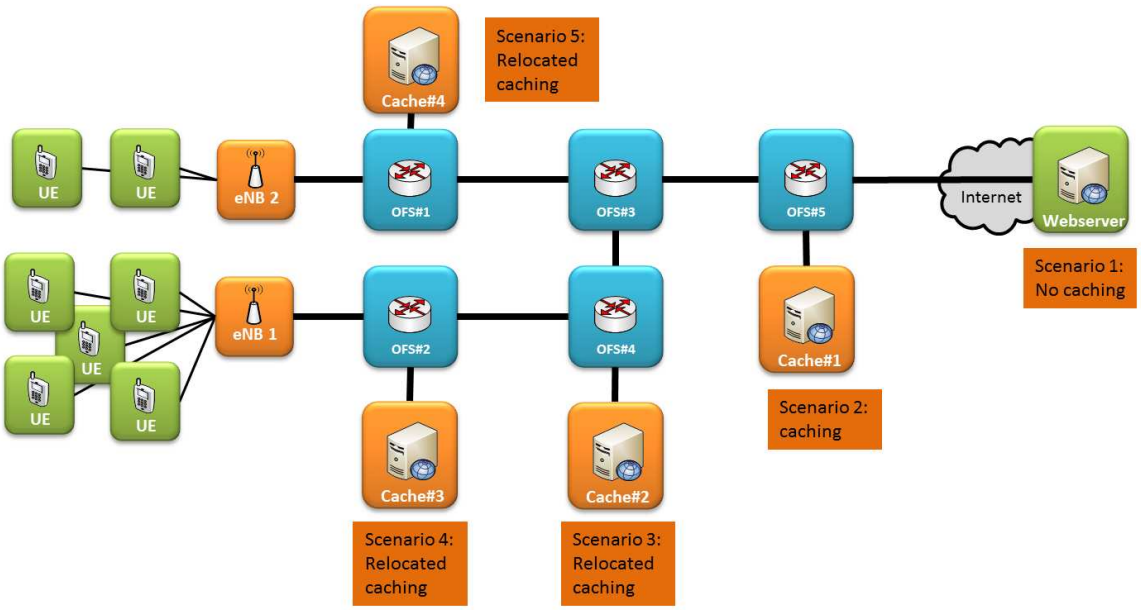

Figure 16: Test network

network to the effect of having different numbers of users in different base stations. The first test case consists into fetching the objects directly from the public Internet outside the mobile network and this case is used as a reference to compare the load generated by the other test cases, when content is cached in different locations as shown in Figure 16 [2].

In order to quantify the impact of our caching system on the network, we consider the sum of the loads of all the links as a network load metric. Thus, a request of 1MB going through 7 links will generate a network load with value of 7MB. This network load metric is used to quantify the reduction of the load in the total network when introducing the caches. This metric also includes the resource consumption required to move contents. We quantify this network load metric as

$$\sum_{i=0}^{N}\sum_{j=0}^{n} x_i * y_{i,j} + \sum_{i=0}^{N}\sum_{j=0}^{n} x_i * z_{i,j}$$

with N being the number of requests issued from all the base stations, n the number of links, x_i being the size of the response, $y_{i,j}$ being 0 if the response does not go through the link, 1 otherwise, $z_{i,j}$ being 1 if the response has been moved between caches through the link, 0 otherwise.

Figure 17 presents the network load reduction depending on the cache location. It is displayed as a percentage of the network load without caching, depending on the repartition of the users on the base stations. As expected, the network load is the same without caching and with a cache located on the same switch as the gateway (scenario 2). But the load reduction increases when the cache is located closer to the

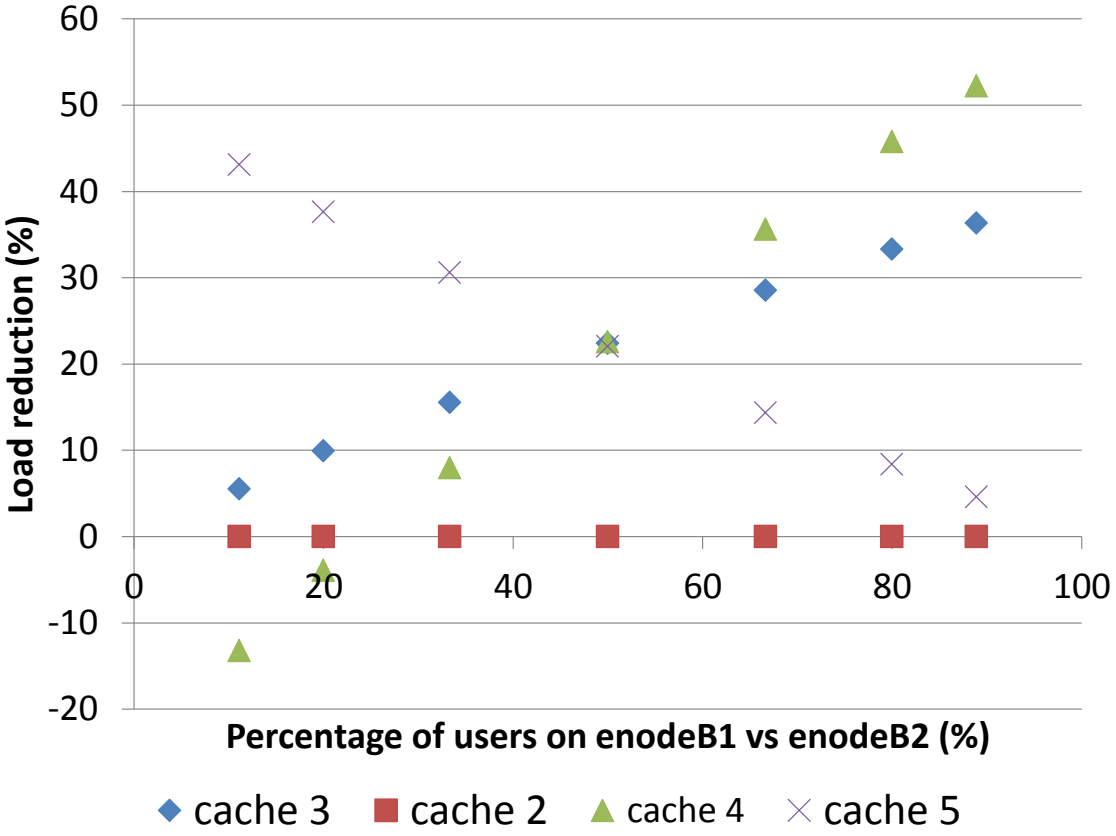

Figure 17: Network load depending on the repartition of 7 users [2]

users. Thus, when all the users are on the first base station, the cache located on the switch of the first base station is reducing the most the load, while the one situated on the switch of the other base station increases it. When all the users are on base station 2 the situation is inverted. The results show that the same cache can achieve both the worst and the best load reduction, since a badly selected location that is not up to date with the users' location and requests, increases the load and degrade the performances of the whole system by increasing the load. This demonstrates the need for short periods in relocation operations, since outdated locations can damage the performances. The cache location should be chosen depending on where most of the users requesting it are usually.

4.5 Issues of the system

The first issue of the system lies in its implementation. This first testbed was implemented using python and we preferred simplicity over performances. For example, the input/output implementation of the cache is not optimal. The system has trouble to handle several clients at the same time. Moreover the caching library has some issues, even if most of them have been fixed in the new release.

Aside of those implementation details, the main problem of this design is that the redirection is based on the destination IP addresses. It means that the granularity

for the choice of the caching location is per IP address, so per website, since having different IP for a same website redirected to different caches would only lead to duplication of the content. So, for example, the finest selection for caching location can only be to redirect the content of Youtube (http://www.youtube.com) to a cache and the content of Facebook to another. But it does not take into account the differences that can exist inside those websites such as the language. For instance, the caching locations can be selected depending on the language of the contents and users, so that it would be closest to the people using this language instead of having a single cache for all the languages. A solution could be to redirect Youtube IP addresses to several caches so that each specific area can access to a closer cache with its dedicated content but then some videos that are the most demanded, such as international hit clips, would be duplicated without any control on this duplication level [2].

Thus we propose some optimizations to the design in order to fix the flaws.

4.6 Summary

In order to validate the concept, we implemented a testbed based on redirection rules matching the destination IP addresses. The simplicity of the system permits minimal resource consumption while improving the performances. However, this simplistic design implies several flaws in our system, such as a problem of granularity in the handling of the contents. This testbed shows that this concept can optimize caching but that it needs high control level for maximizing the performance increases. Thus in the next section, we are focusing on improvements to the system for a better control over the contents.

5 Caching integration in Software-Defined networks

Along the thesis work, several solutions were proposed in order to improve the first implementation. Three solutions will be described and analyzed in this section.

The rules presented hereafter do not include the usual forwarding or routing rules. They are more specific rules with higher priority to match only the desired part of the traffic.

If there are some non-Openflow switches in the network, the solution, when the destination IP and Mac address have been modified, is to change them to the next Openflow switch IP and Mac address. Then this switch will set back the original IP and Mac address. This solution would have a cost in terms of number of rules on the SDN switches around the non-SDN island. Another solution would be to use a GRE tunnel between those switches. The tunneling would have some costs in terms of bandwidth, due to the overhead of the encapsulation, and in terms of processing needed to encapsulate. To adapt to a network with such a non-SDN island, the number of border SDN nodes involved and the size of the island should be considered to determine the optimal solution.

In the different subsections, we will present the three solutions proposed and discuss the different performance criteria. We will then present the setup of the simulation based on those criteria and the results of the simulation.

5.1 Forwarding-on content solution

In ICN networks, the nodes retrieve the contents based on their name instead of their location. The client, the proxy in our case, is not aware of the content location. In order to perform such a service, we use the destination IP field to identify the content like the ContentFlow solution [23].

Figure 18 presents the process of the UE request. The mobile sends a request for a piece of content. This request is redirected to the proxy that will analyze it and query the cache controller, or its own mapping cache for the cache controller responses if the content was already mapped, to get the correspondence between the content and the IP identifying it. The cache controller will also communicate with the SDN controller to set up rules to forward those requests on all the switches along the path. The proxy will then forward the request using the IP-like identifier as the destination IP. The switches will then forward the request based on the destination IP identifying the content, to its location. The cache will then send the response using the IP address of the proxy and usual routing or forwarding.

Figure 19 presents the relocation process. When relocating content, the cache controller will first take care of the transfer between the two caches (one copy in each). When it will be done, it will update the rules on the switch and delete the original copy.

Advantages and drawbacks This solution presents several features and advantages:

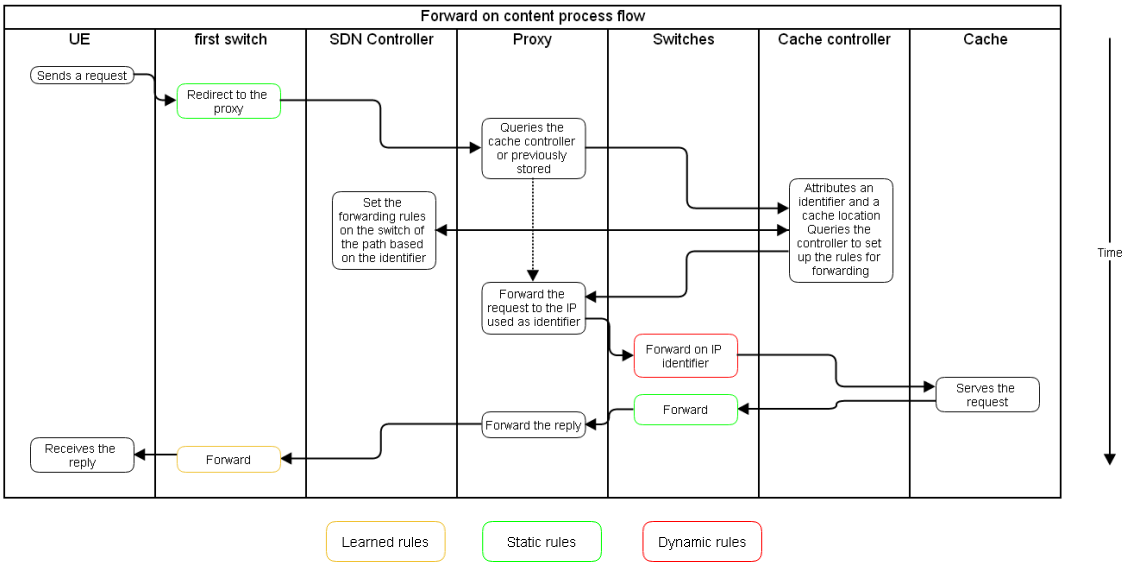

Figure 18: Process of the request for the SDN-based solution

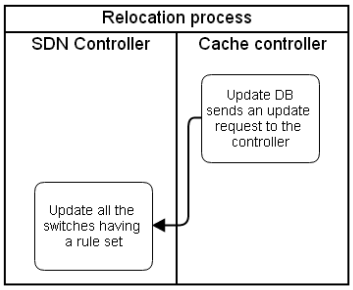

Figure 19: Relocation process for the SDN-based solution

- The granularity for the content location is fine-grained, since each content is identified uniquely; it can be fetched independently from any of the cache.
- The forwarding can be aggregated on the switch per content, independently of the source of the request.
- The path can be modified in order to distribute the load and adapt to the network situation. The path can thus be computed according to the current network load.
- The cache relocation is done by changing the forwarding rules. Instead of reconfiguring every proxy when relocating an object, even from one location to its closest neighbor location, we would only need to reconfigure the closest switches of the caches concerned by the relocation. For example, relocating the content from one node to its closest neighbor node would imply the neighbor switches only; while it would imply as many operations on the proxies as there are base stations where the users are accessing this content, if the proxies

were fetching the contents by their location. Thus, the performance of this proposition depends on the number of hops between the nodes and neighbors involved in the relocation.

However, this proposal has some of the most sever drawbacks of the proposed solutions:

- The number of content that can be uniquely identified is limited by the size of the IP field. This is a problem for IPv4. In IPv6, this issue is discarded regarding to the load on the switches and on the controller.

- The system needs a rule per content on each switch of the path. If all the IP pool (IPv4) was completely mapped to contents, there would be 2^{32} rules for forwarding content on a switch where all the mapped contents are going through. Considering the current switches, this is obviously not possible. This load repartition can't be easily improved since the users are fetching the content independently to each other's, even if some patterns can be found. Moreover, when using IPv6 to solve the problem of the limited number of contents identifiable, the problem of the number of rules gets worst, due to the high number of possible rules.

- The relocation implies to update all the switches that are part of the path between all the users requesting an object and the cache providing it. In case of an object that is requested by at least one user on each enodeB, a high number of switches may have to be updated. There might be more operations needed to do it than to update all the proxies with the content location. Thus the load on the SDN system would be more important than the one on the caching system for updating the proxies.

- The load on the controller, on the links between the controller and the switches, and on the switches themselves is too important. Every time a new cached piece of content is requested in a proxy, or the network situation evolves, leading to a relocation, the controller will have to set new rules or update the rules on a high number of switches. Thus maybe saturating the links or the controller. There would thus not be any bandwidth or computation power left for any other application. For example, in a simple network such as a tree network, if users are requesting a new piece of content in half of the base stations, this would lead to nearly twice as much operations of switch configuration, while if the content was fetched by its location, it would not lead to any operation more since the proxy is already retrieving the content identifier from the cache controller. It would instead retrieve the location of the content. Initially, it is more expensive to set the rules in the switch than configuring the proxy.

- If some switches do not support SDN, they will not be able to forward the packets, or they would forward it based on wrong routing tables or forwarding rules. Thus, the rules on the last Openflow switch must be adapted so that the

destination mac address is changed to the next openflow switch in the path so that the packets can be forwarded to it. If the packet cannot be forwarded but must be routed, solutions such as GRE tunneling should be used, but it adds some limitations such as a new header and bigger bandwidth consumption. The payload may also need to be fragmented, implying a higher load on the endpoints of the tunnel. Thus some adaptation as discussed above is needed to take into account the non-SDN zone.

- The relocation implies a break in the connection during the transfer going on. Thus, an overhead of a TCP RST and a TCP handshake exists.

Rules needed This solution requires three kind of rules on the Openflow switches:

- A first static rule is needed per base station to redirect the http traffic to the proxy.
- A rule per switch on the path of all contents. It can be up to 2^{32} rules per switch.
- A single static rule per cache to terminate the forwarding-on-content-name. It can also be split into two separate rules.

Table 20 presents the rules in the different switches. The first switch after the UE and the last switch before the UE are the same switch, idem for the one at the cache level.

Runtime operations on the SDN controller This architecture cannot run without a SDN controller. The interactions with the SDN network are the central part of this design. The SDN controller is a shared resource among all the systems, and it is an essential component of the network. Thus, the interactions must be thought about carefully, in order not to overload the controller by adding the individual loads of the different systems. The SDN controller being shared, its operations will be more expensive than any other one in the service architecture. That is the reason why we focused on the number of runtime operations needed:

- Setting a new path to a new piece of content from a proxy: the number of hops between Openflow switches that are not already configured for the object, will be the number of operations performed by the controller on the switch.
- Relocating content: There will be an operation per switch on which a forwarding rule is updated, added or deleted.

Caching system requirements Aside of the controller resource, the system will also have some requirements, on the knowledge needed, or on the hardware :

- Full knowledge about the whole network architecture
- Full knowledge of the content of each cache

Switch	Match	Action	Aim
First after the UE	TCP, port 80	Set dst IP and Mac to proxy, output, learn [match (TCP, dst IP = src IP, dst port = src port, input port = output), action set src IP, set src Mac]	Redirect to the proxy, learn a rule to set back the original IP based on the src IP and port
All between proxy and cache	Dst IP, TCP, port X	Output	Forward on content identifier
Last before the cache	Dst IP, TCP, port X	Set dst IP and Mac to cache, set port 80, output, learn [match (TCP, dst IP = src IP, dst port = src port, input port = output), action set src IP, set src Mac]	Forward to the cache, learn a rule to set back the original src IP and Mac address of the reply
First after the cache	Dst Mac, IP and port, input port	Set src IP, Mac and port	Set the content identifier as source IP (learned rule)
Last before the UE	Dst Mac, IP and port, input port	Set src IP, Mac and port	Set the webserver IP as src IP (learned rule)

Figure 20: rules on the SDN switches for solution 1

- Stateful knowledge about the rules set for the path (memory requirements)
- Fast computation of the best location and shortest path (computation requirements)

5.2 Proxy fully-based solution

The architecture would be similar to the one of the previous solution, but it would not use any forwarding-on-content, and the cache would be independent of the SDN nodes. Figure 21 presents the process of the mobile's request. The proxy would fetch the content using its location such as the IP address of the cache where it is or will be stored.

Figure 22 presents the relocation process for this solution. When relocating the content, the cache controller will push the update to all the proxies that have

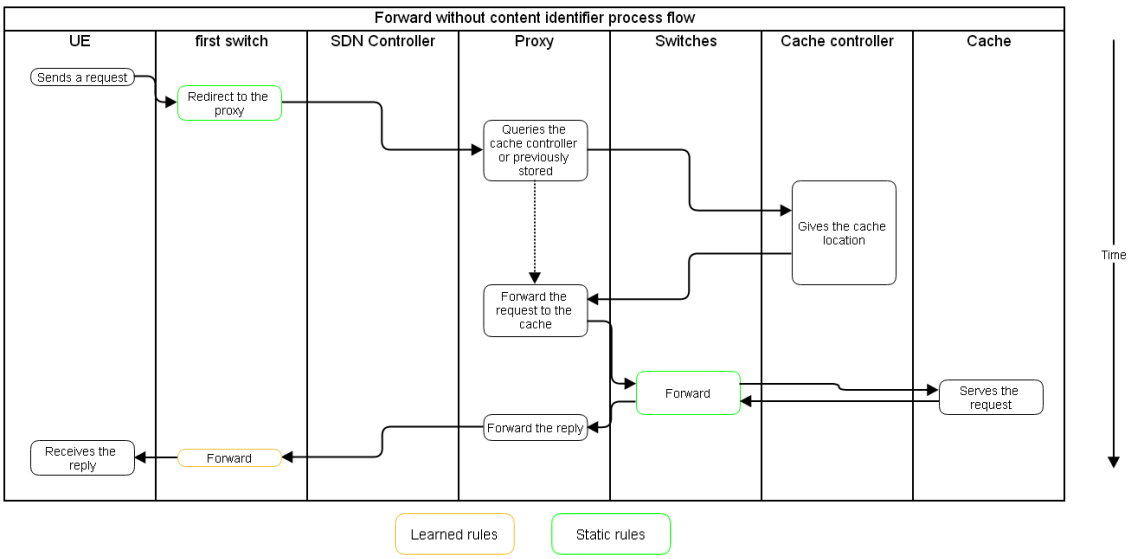

Figure 21: Process of the request for the proxy-based solution

requested the content in a given time interval. The cache controller would first trigger a copy of the object from a cache to another and then push the change in the mapping of the URL and the locations to the proxies. The original cache would then reply with "301 Moved permanently" if the client proxy connects again for the object. The current download from the original cache would be completed before the removal of the object from the cache storage. This solution does not interact with the SDN controller and thus is closer to the usual caching systems not using SDN.

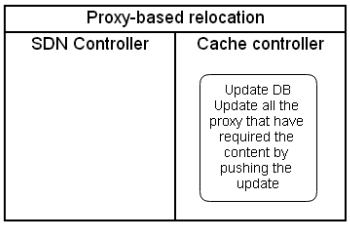

Figure 22: Relocation process for the proxy-based solution

Advantages and drawbacks Compared with the previous solution, this solution has some advantages:

- The system uses only static rules to redirect the client request to the proxy. There is no other use of SDN controller to set up rules. Thus, the load on the controller is restricted to the normal operations for having a functional network. Moreover, those operations are required on any Software-Defined

Network, thus the footprint of the caching system on the SDN controller is highly reduced.

- When accessing a new content from a proxy, the location is retrieved from the cache controller so it is faster than setting the rules on all the switches of the path. The load on the cache system is the same since this retrieval was already implemented in the previous solution.
- Relocation is more efficient for a piece of content that is accessed from most points of the network and that is moved far from the original cache because it implies less operation.

The main drawback is that relocating is more expensive for small moves when it is accessed from most points of the network. When several users on different base station are accessing an object, several proxies are involved in fetching the content. Each of them has a mapping of the URL with the cache location. Thus all those proxies must be updated, leading to the same number of operations. If the move is limited, only the direct neighbor switch will need an update in the previous solution. That might reduce the number of operations, but their cost depends on the load already existing on the SDN controller.

Rules needed A first static rule is needed per base station to redirect the http traffic to the proxy. The rules are detailed in Table 23.

Switch	Match	Action	Aim
First after the UE	TCP, port 80	Set dst IP and Mac to proxy, output, learn [match (TCP, dst IP = src IP, dst port = src port, input port = output), action set src IP, set src Mac]	Redirect to the proxy, learn a rule to set back the original IP based on the src IP and port
Last before the UE	Dst Mac, IP and port, input port	Set src IP, Mac and port	Set the webserver IP as src IP (learned rule)

Figure 23: rules on the SDN switches for the Proxy-based solution

Caching system requirements This solution does not imply any operation at runtime on the controller, other than setting the static rules at switch connections. But it has similar requirements :

- Full knowledge about the whole network

- Full knowledge of the content of each cache
- Full knowledge of the requests processed in the proxies during a defined time for pushing the updates
- A single Openflow switch per base station to redirect to the proxy. They can be pooled.

As a whole, the requirements for the caching system would be the same since the amount of data and needed computations are slightly similar. The only difference with the previous solution in terms of load is that the load of the caching system on the SDN controller in the previous solution will be transferred to the cache controller.

5.3 Hybrid solution

This solution is a merge of the two previously presented, in order to improve its efficiency and reduce the load. It uses a modified version of the forward-on-content principle. The pool of identifiers is divided into several smaller pools, every cache having its dedicated pool. These pools can be subdivided to separate the content types. For example, the n first bytes indicate the cache, the y next bytes indicate the type of the content and the remaining bytes identify the content. Thus, the identifier is unique per cache and per content type. For instance, if we use duplication, on different caches, the same content would be identified differently. [2]

In this solution, the rules can be aggregated per cache. Hence, there would be static rules to forward the packets to each cache, on all the switches, using masks to match only the bytes identifying the cache of the content identifier. A single rule is needed to match all the packets for a cache, instead of one rule per object as proposed in the SDN-based solution.

Figure 24 presents the process of the mobile's request. When retrieving a new object, a proxy would be given the identifier of the content that includes the cache where it will be located. The proxy would thus be able to fetch it without installing new rules since the rules matching the content identifier on the bytes identifying the cache are already installed (static rules).

Figure 25 presents the relocation process for this solution. The relocation could be performed in two ways. The first one would consist in changing the mappings in the cache controller, without pushing it to the proxies, and setting rules with higher priority in the switches to redirect this content identifier to the new cache location, overriding the rules matching with masks for these identifiers. The rules would exist only for the remaining time the mappings are cached in the proxies. Or the relocation could be performed by pushing the update to all the proxies needing it. The decision on which method should be used shall be based on the number of operations required and the load it would induce on the different services, so on the efficiency of the operations. It could also be based on some parameters given by the users in order to promote a method over the other [2].

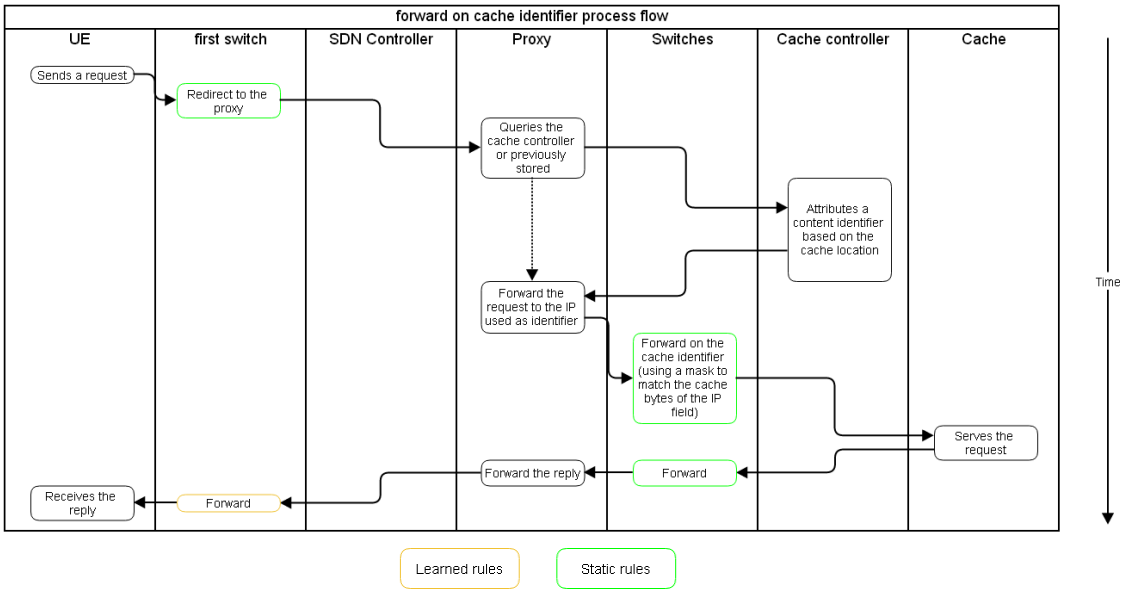

Figure 24: Process of the request for the hybrid solution [2]

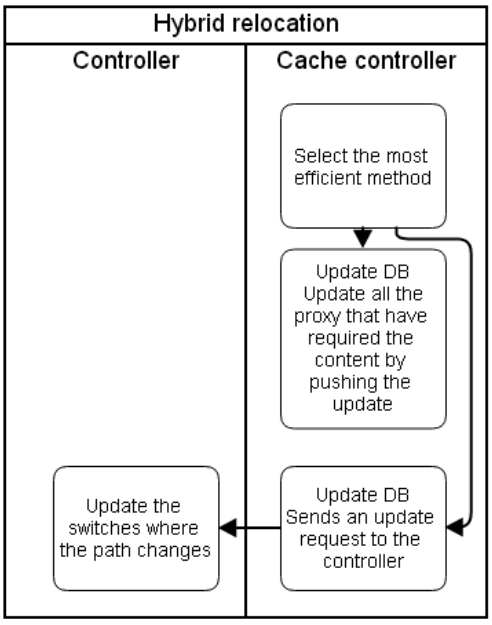

Figure 25: Relocation process for the hybrid solution

Advantages This solution presents some advantages from both previous solutions :

- This system reduces the time needed to perform the relocation operation by choosing the most efficient method.
- It reduces the overall load on the whole system by increasing the efficiency.
- The parameters could be tunable in order to promote one technique over the other. This has been designed to best fit the reality.

But it also presents some drawbacks, either from the previous versions or specific to this new version. The first drawback is that more computation and time are needed to determine what would be the most efficient way to relocate. This should be estimated more precisely in some further work, depending on the relocation triggering algorithm. Moreover, some of the drawbacks of the first version apply here, such as the problem of the non-SDN switches islands. They could be integrated using the proposed technique, changing the mac address to the next-SDN-hop or tunneling the content.

Rules needed In this solution, several rules would be needed :

- A first static rule is needed per base station to redirect the http traffic to the proxy.
- Static rules, one per cache per switch to forward to the cache on each switch using masks to match the identifiers.
- One rule per redirection per switch where it is needed, with a timeout slightly longer than the time the cache controller reply is cached.

Table 26 presents the rules used for this solution.

Caching system requirements As runtime operations, the SDN controller will have to set redirection rules in some of the relocation cases. The system will have more requirements :

- Full knowledge about the whole network
- Full knowledge of the content of each cache
- Stateful knowledge about the rules set for the path (memory requirements)
- Fast computation of the best location and shortest path (computation requirements), always needed to compare the cost of the operations
- Full knowledge of the requests processed for a defined time for pushing the updates
- Unspecified number of SDN switches, at least as many as the proxy-based solution.

Switch	Match	Action	Aim
First after the UE	TCP, port 80	Set dst IP and Mac to proxy, output, learn [match (TCP, dst IP = src IP, dst port = src port, input port = output), action set src IP, set src Mac]	Redirect to the proxy, learn a rule to set back the original IP based on the src IP and port
All between proxy and cache	Dst IP (with mask), TCP, port X	Output	Forward to the cache
Last before the cache	Dst IP, TCP, port X	Set dst IP and Mac to cache, set port 80, output, learn [match (TCP, dst IP = src IP, dst port = src port, input port = output), action set src IP, set src Mac]	Forward to the cache, learn a rule to set back the original src IP and Mac address of the reply
First after the cache	Dst Mac, IP and port, input port	Set src IP, Mac and port	Set the content identifier as source IP (learned rule)
Last before the UE	Dst Mac, IP and port, input port	Set src IP, Mac and port	Set the webserver IP as src IP (learned rule)
Any	Dst IP, TCP, port X	output	Redirection rule (higher priority)

Figure 26: rules on the SDN switches for the hybrid solution

5.4 Simulation setup

We set up a simulation to compare the different solutions. The simulation is presented and analyzed in this section.

5.4.1 Description of the model

In order to determine the impact of those solutions, we have set up a simulation. We simulate a network of 16 base stations on which we randomly distribute users, up to 10 per base station, with 31 switches, all of them SDN nodes with caching capabilities. The network is pyramidal and we attribute costs to the links increasing when getting closer to the edge. Figure 27 presents the simulated network. [2]

Let the time be discrete, the requests be unrelated to each other and N be the number of objects on the internet, ranked by popularity order. The first object is the most popular and the last is the least. Let all the users send a request at every time slot. Let a be the identifier of the base station and x_a be the number of users on the base station a. Let t be the interval between relocations. [2]

In these simulations, we are defining the time unit as time slots, to be as close to the reality as possible. A time slot is a variable unit of time that can be changed to best fit the reality. We define the time slot as the average time between two requests from the same user. Hence, this is highly dependent on the behavior of the users and on the time of the day. For example, the biggest traffic is between 8pm and 0am [12]. So at that time, the time slots are shorter than from 1am to 6am. Thus, the simulation is not dependent of a specific users' behavior but can be adapted to fit the demand distribution.

We assume the requests of the users follow a Zipf-like distribution: let i be the position in the popularity list, the probability that the user fetches the object i is

$$P_{N,\alpha}(i) = \frac{1}{\sum_{j=1}^{N} \frac{i^\alpha}{j^\alpha}}$$

The value of α varying for each network, we use fixed values from the set {0,65; 0,7; 0,75; 0,8; 0,85; 0,9; 0,95; 1} to study its impact and then we arbitrarily set this variable to 0.9 [31] in the second set of simulations [2]. We limit the number of existing contents that can be fetched by the users to 50000 items such as web pages or images, and we consider that the average size of the responses is 1MB, considering the repartition of size of the requests on the web [26].

Figure 27 shows the network simulated. The base stations are numbered from left to right, 1 being the most left. The switches are numbered from top to bottom and left to right, 1 being the top one, 2 the left one on second row, 3 the right one on second row, 16 being the leftmost on the last raw and 31 the rightmost.

We run two types of simulation. Both are using a simple relocation decision algorithm. If the demand for an object has evolved, the algorithm computes the new optimal location, minimizing the network load. Then, by comparing the previous

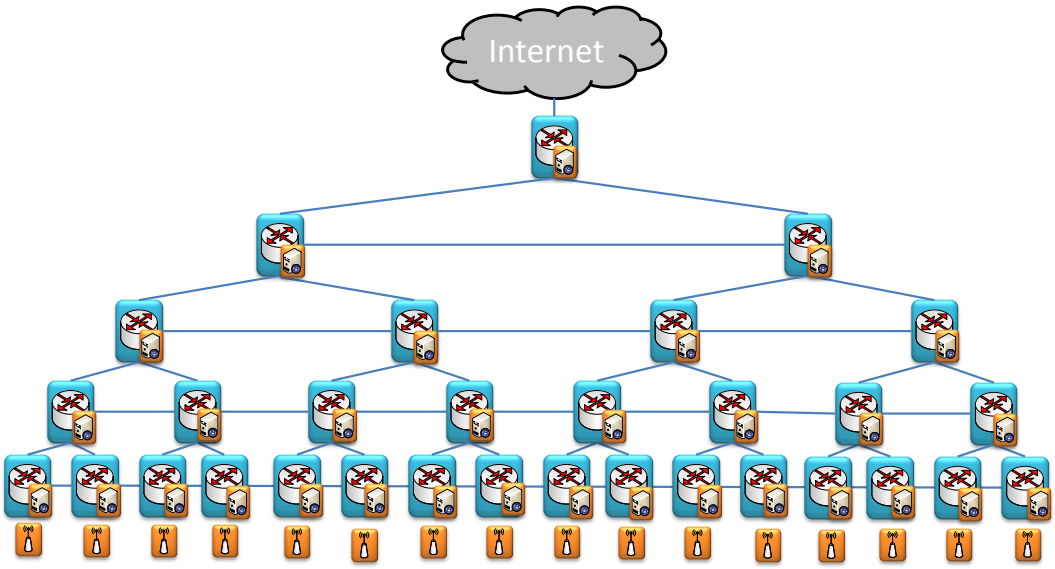

Figure 27: Simulation Network

EnodeB	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number	6	9	5	7	2	3	6	4	10	6	7	5	4	3	9	1

Figure 28: Repartition of the users

location of the object and the computed one, the algorithm determines if a relocation is needed. It compares the sum of the costs of fetching the object by all the users downloading it at the current location, including the new request, with the same sum for all the other locations, adding the cost to move the object between the caches. If the lowest sum is not the current one, it proceeds to the relocation. This algorithm is a starting point and will be improved in future work. The first type of simulation runs this algorithm after each new request. We set up a warming period during which the users' requests are gathered to create the original pool. Afterwards, we study the properties of the next requests. The second type of simulation runs the algorithm based on time. Each set of relocations is separated by a time interval. The requests issued by the users are gathered during those intervals and used to compute the new locations of the contents.

For the first simulations, we run the program with $5 * 10^4$ contents, and a warm-up equivalent to 240 requests per user. The users are distributed randomly on the base station, as shown in Table 28.

The requests are simulated by drawing for each a value from the Zipf-like distribution. Then the current location of the cache for each content is calculated to minimize the sum of the weight of the path to all the users requesting the content. Then the simulation checks if a new request for a piece of content on a base station leads to a relocation or not by using the relocation algorithm.

Considering $x_{a,i}$ the number of users on the base station a requesting the object i, and assuming that the entire infrastructure is known beforehand, we can build a

matrix giving the weight of the shortest path between every base station and caches such as:

$$M_{BS,C} = \begin{pmatrix} X_{BS1,C1} & X_{BS1,C2} & \cdots & X_{BS1,Cm} \\ X_{BS2,C1} & X_{BS2,C2} & \cdots & X_{BS2,Cm} \\ \vdots & \vdots & \ddots & \vdots \\ X_{BSn,C1} & X_{BSn,C2} & \cdots & X_{BSn,Cm} \end{pmatrix} = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{pmatrix}$$

The optimal cache location j would be the one minimizing

$$\sum_{i=0}^{N} x_{i,j} * y_i + z_{o,j}$$

with N being the number of base stations, n being the number of links, x_i being the weight of the path between the base stations and the cache j, y_i being the number of requests on the base station, $z_{o,j}$ being the weight of the link between the current location o and the optimal location j.

5.4.2 Limitations

This model has several limitations [2]:

- The model does not take into account the relationship between the requests such as the CSS sheet, the sprites and icons after downloading the main web page.
- The simulation does not allow comparing the first solution implemented to the other one since the proof-of-concept design was based on a pooling of the requests per website and this model does not take these relationships into account. Moreover, loading a single webpage induces loading several contents from other websites, such as google advertisements service. The relation-ship between the content drawn from the Zipf-like distribution and the webservers reached cannot be established in this model.
- Every request is going from the proxy through the cache, even if the response is not cacheable. We didn't include the non-cacheable items in our model.
- The selection of different parameters for the zipf-like distribution is not reflecting reality since the value of α is extremely dependent of the network parameters, such as the type of users etc.
- The small number of contents requested used in the example modifies the probability law.
- We do not consider the need for computation power. We infer that we have a fitted machine to run the services of the cache controller

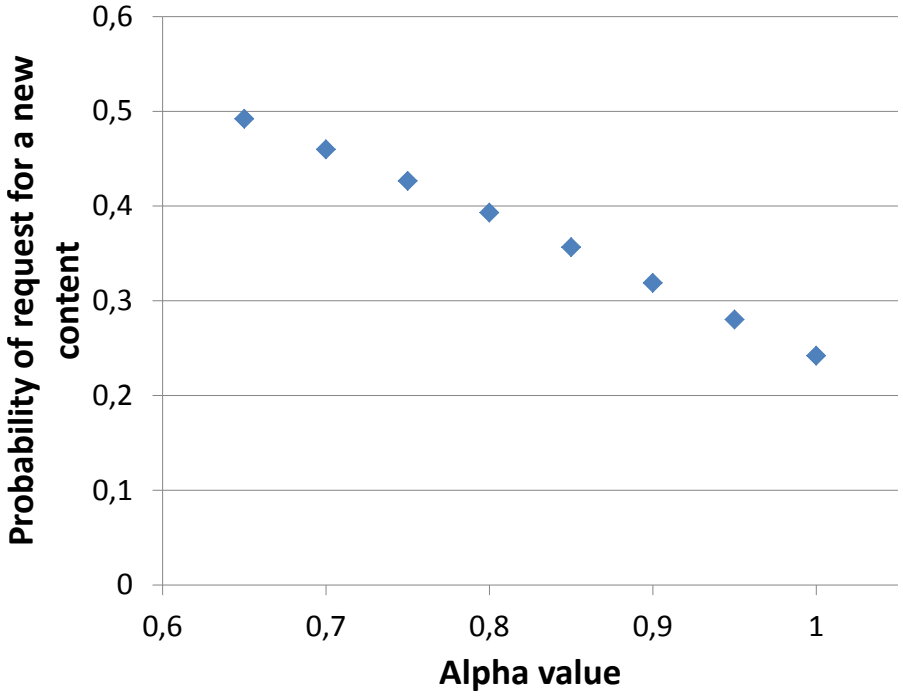

Figure 29: Probability of a request for a not yet requested content

5.4.3 Results of the first simulation set

First, we focused on the probability that the request will be for a new piece of content, depending on the value of α. Figure 29 presents the results of this simulation.

The α parameter has a strong impact. The diversity of contents requested is directly linked to this value. When α is close to 1, the best-ranked contents have more probabilities to be requested than when α is close to smaller values such as 0,7. Thus, the contents requested are more diverse with lower values of α than with values closer to 1. The probability of requesting a new content is higher with low values of α since the probability of getting more diverse contents is higher. The probability of fetching a new piece of content is also related to the time interval during which we gather the requests since it limits the number of contents requested in comparison with the available amount. Thus the probability of requesting an unrequested object is higher when the interval is lower.

Then we focus on the relocation probability for the three solutions proposed. Figure 30 presents the relocation probability when drawing a new content for a base station, taking the content probability into account. The relocation probability is depending on the value of α. When the content is diverse, the heavy tail of the distribution is more important, thus, there is more randomness during the drawing. Hence, the content location varies more and the relocation probability is higher. This probability is high due to the simple relocation algorithm but since it is the case for all the solutions, it does not interfere with the comparison. However, the impact of α should not be neglected since, by adding randomness in the drawings for the small values, it adds randomness in the moves of the contents. The moves

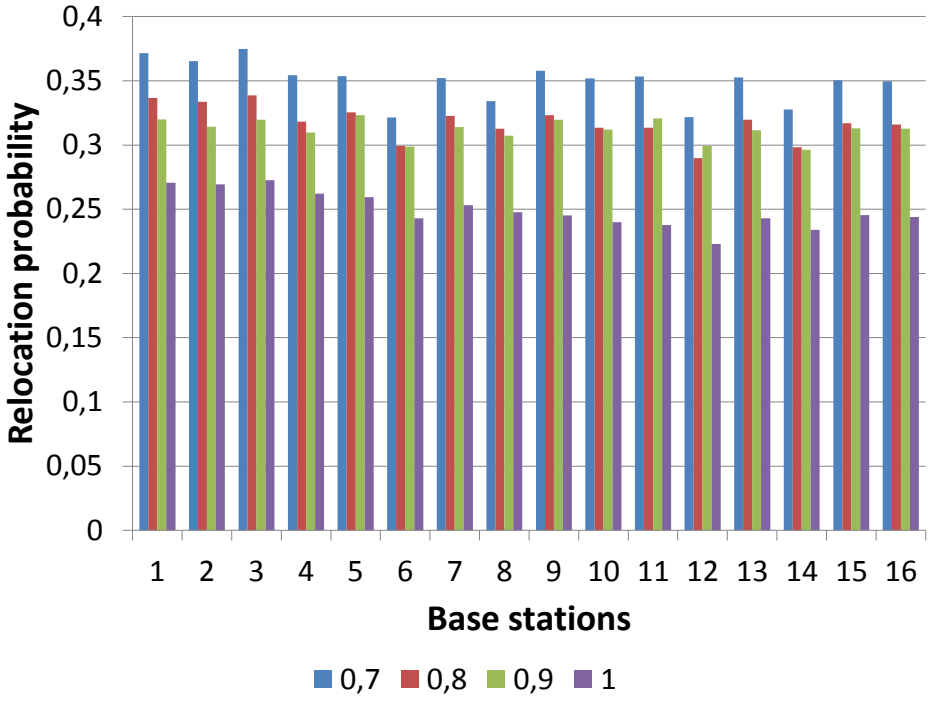

Figure 30: Probability of relocation depending on α

are thus less localized. Afterwards, we focused on the number of operations needed to update the cache locations.

Figure 31 presents the average number of operations needed for every new request for a value of α of 0,65. The SDN-based solution requires twice as many operations as any of the two other solutions. This is due to the forwarding on content that requires many more configurations (on every switch of the path) compared with the configuration of the proxies since there are usually less proxies to configure than switches.

Then Figure 32 presents the repartition between the configurations of proxies and switches in the hybrid solution. Most of the relocation operations concern the proxies but in some cases, it is more efficient to configure the switches thus some of the operations are performed on the switches and not on the proxies. This can be induced by comparing the proxy-based solution that uses only proxies configuration and the hybrid solution that mainly uses proxies configuration but also some switches configuration in Figure 31.

Figure 33 and Figure 34 presents the same set of results for $\alpha = 1$. In this case, the average number of operations is smaller because the probability of relocation is smaller due to the value of α. And the average number of operations for the proxy-based solution (proxy configuration) is relatively higher compared to the other solutions. This is because the contents requested are less diverse, so the contents are requested from more base station, thus there are more proxies to configure than for a lower value of α. As a consequence, the proportion of switch configuration increases in the hybrid solution.

With regard to these figures, the SDN-based solution presents a higher average

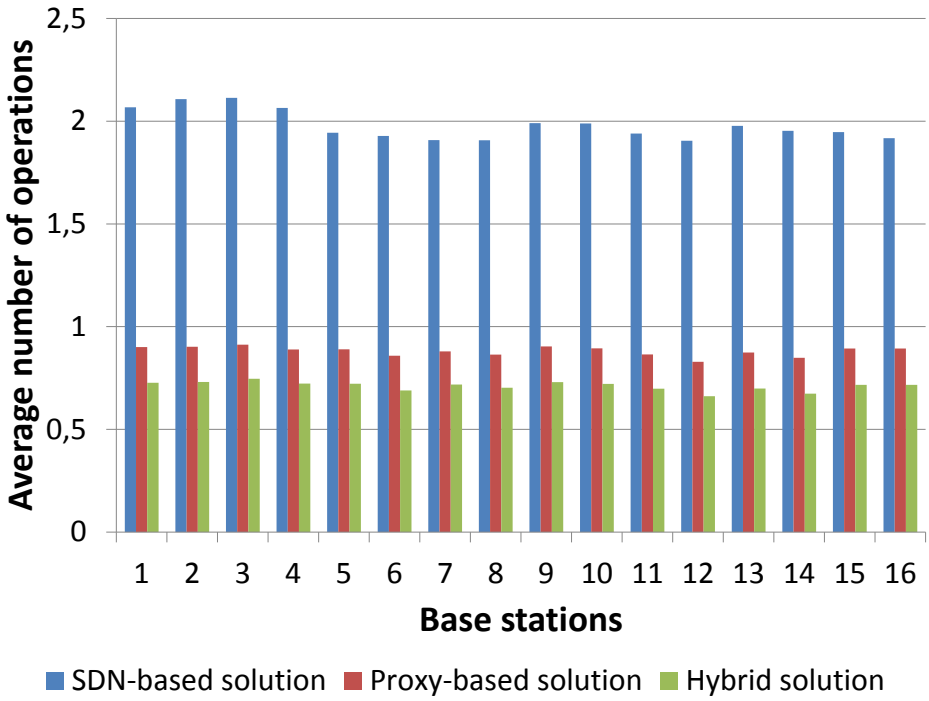

Figure 31: Average number of operations ($\alpha = 0,65$)

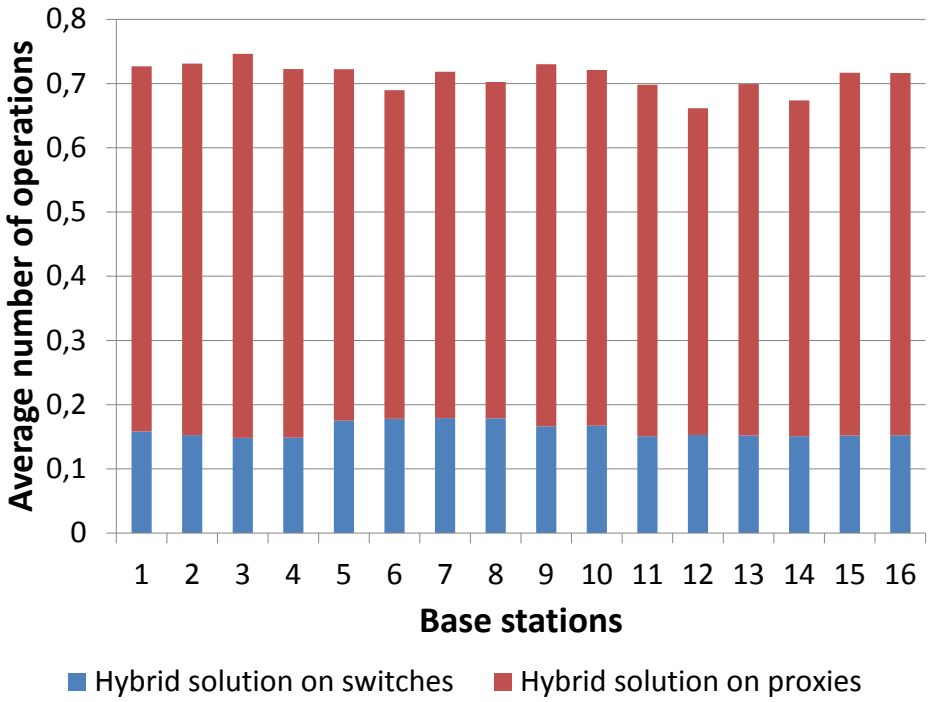

Figure 32: Average number of operations in the hybrid solution ($\alpha = 0,65$)

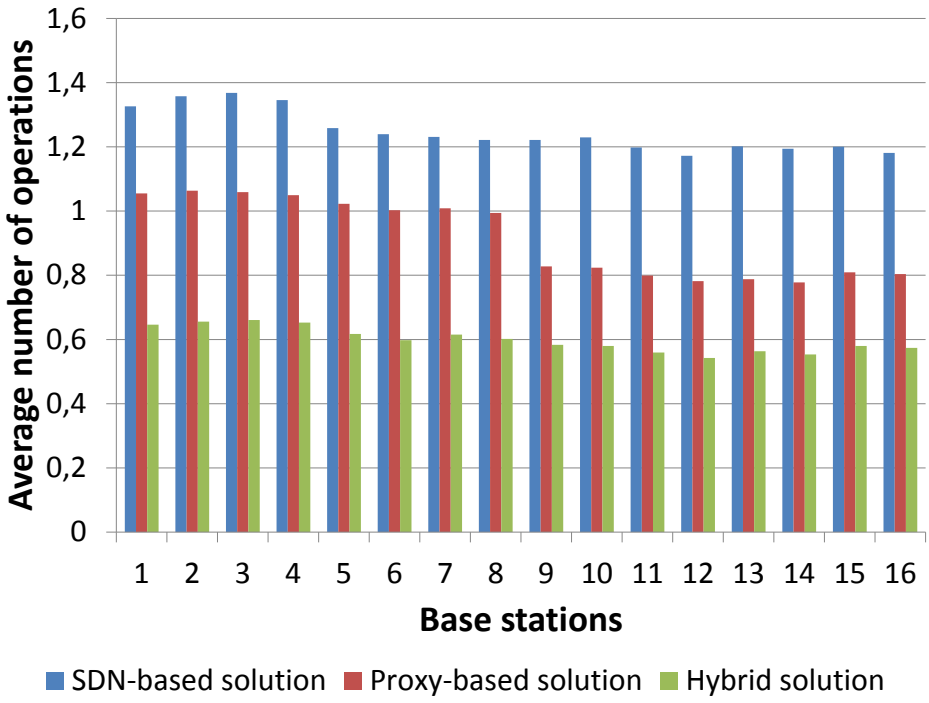

Figure 33: Average number of operations ($\alpha = 1$)

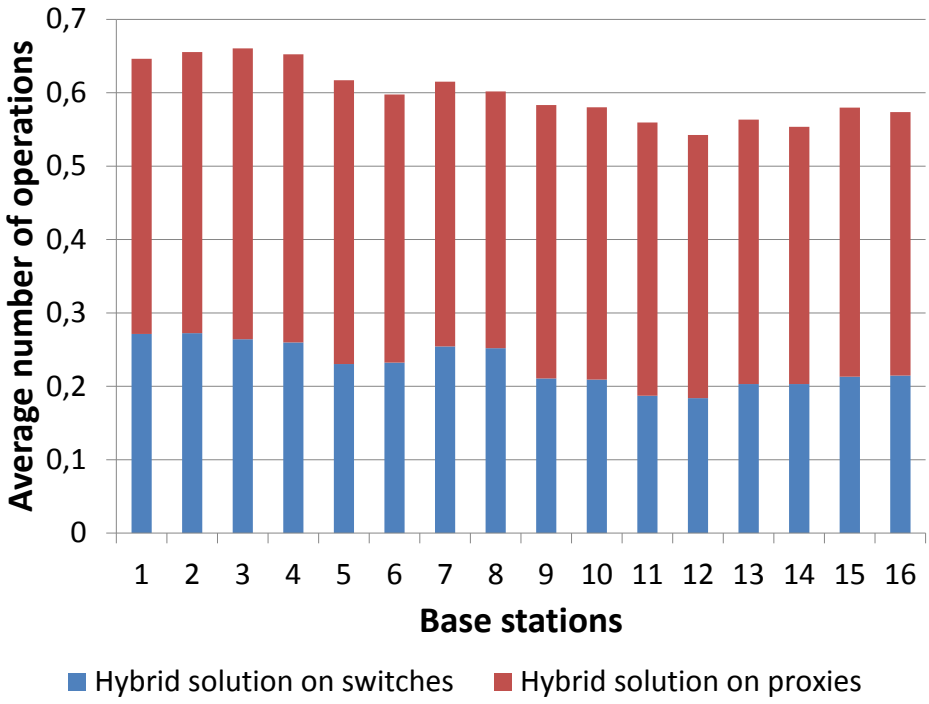

Figure 34: Average number of operations in the hybrid solution ($\alpha = 1$)

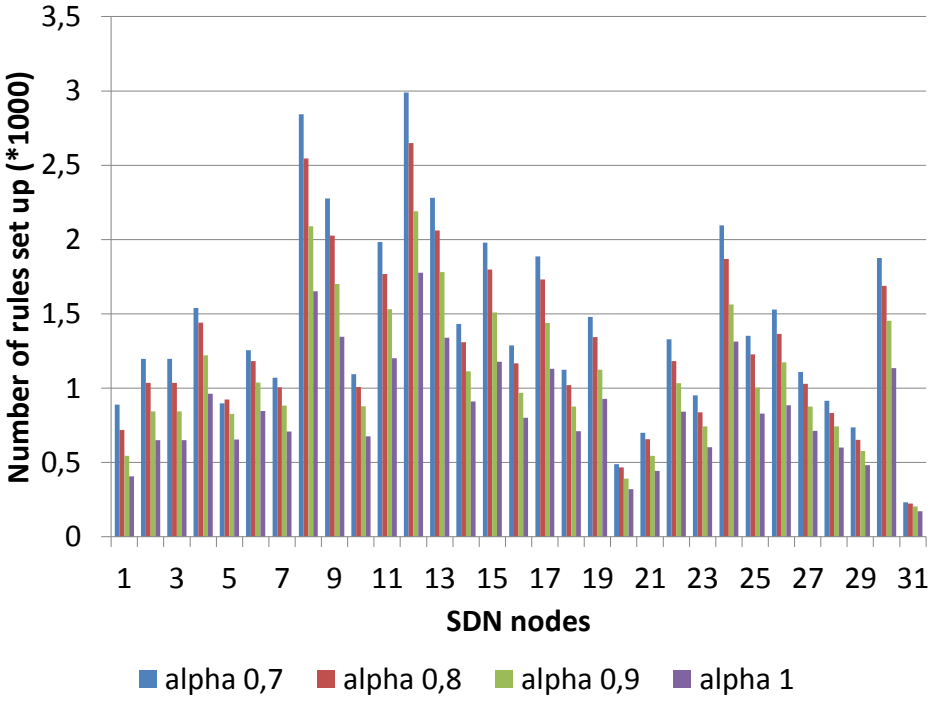

Figure 35: Number of rules on the switches for the SDN-based solution

number of operations whatever the value of α is. This solution does not perform as well as the two other and may not be scalable due to its feature of forwarding-on-content that implies a rule for every contents on every switch of the path. In order to check the scalability, Figure 35 presents the number of rules on the switches for this solution. Depending on α, the highest number of rules is between 1750 and 3000 for 50000 contents and a gathering interval of 240 time-slots. Considering the number of contents on internet and that the timeout would be longer in order to reduce the number of operations, the number of rules is not scalable. This solution presents too severe drawbacks to be implemented. Since the hybrid solution is performing better in the simulation than the proxy-based solution, we will focus on its implementation as our pilot. So we change the underlying mechanism from the IP-based redirection that was implemented first in the testbed to the hybrid solution to check whether the concept could be applied.

We now focus on the relocation triggering method. As displayed by Figure 31 and Figure 33, the average number of operations per request is too high. This relocation method is not scalable. Thus we have run a second set of simulations with a different relocation triggering method.

5.4.4 Results of the second simulation set

In these simulations, the relocation method is based on the same algorithm but is not triggered for each new request. Instead, it is based on time intervals. We define a time interval between each set of relocations. Thus, we study the impact of this time interval on the network load. The simulation includes a warm-up period for

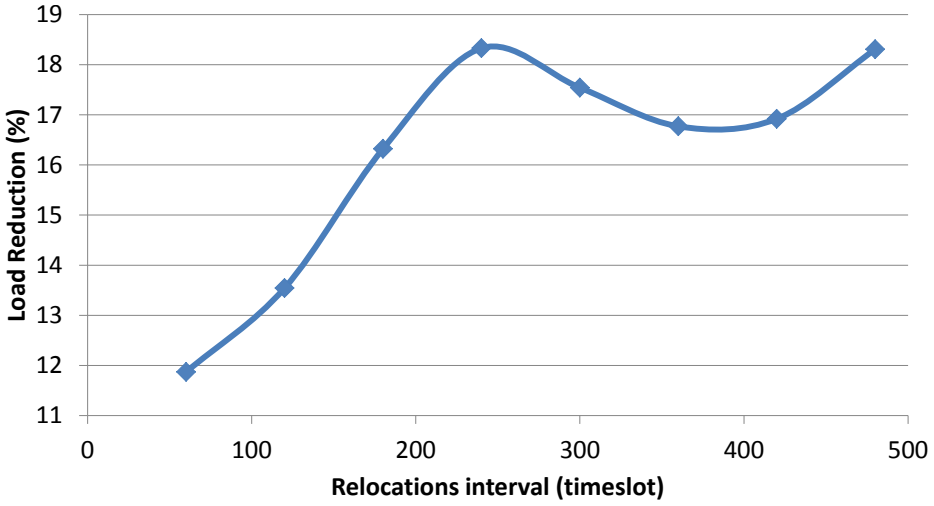

Figure 36: Network Load Reduction. [2]

gathering the requests and creating an initial pool. The location of the contents is computed first from this pool. Then the drawings continue during the time interval. The locations are again computed, using the new requests pool. Finally, the two sets of locations are compared to determine if some contents must be relocated.

Using the network load metric discussed earlier, Figure 36 presents the load reduction quantifier depending on the relocations interval.

The load reduction presents three distinct phases. The first phase is increasing the efficiency due mainly to the reduction of the number of relocation which leads to reduced bandwidth consumption for the relocations. The number of requests also increases and is thus more representative of the whole behavior of the clients for the most requested contents, which leads to a better placement. In the second phase, the efficiency is decreasing due to the lack of reactivity of the system. This in practice means that when the period between relocations is high the users request content that is not properly allocated. This phase is a transition between the state where we prefer a reactive system that detects small changes in the user's requests and is adapting to it compared with a system relying more on the trends as used in the third phase [2]. The network load reduction increases again and we consider the reason is that users are fetching the same content which has been reallocated properly and does not need to be changed. In that case, the system is able to find a pattern in the requests of the users and use it to place the contents optimally.

We focused then on the different relocation methods permitted by our system. Considering that updating a proxy or a switch is a single operation, we compare the number of operations needed for each method. Figure 37 displays the number of relocation operations per period of relocation and shows that the hybrid method is the most efficient. We figured out that the relocation method not using SDN performs slightly better with small relocation periods but far worst with larger relocation periods than the method using only SDN. The high relocation period induces many requests from most base stations on most of the contents, making it more expensive to reconfigure than to configure the switches in general. Those

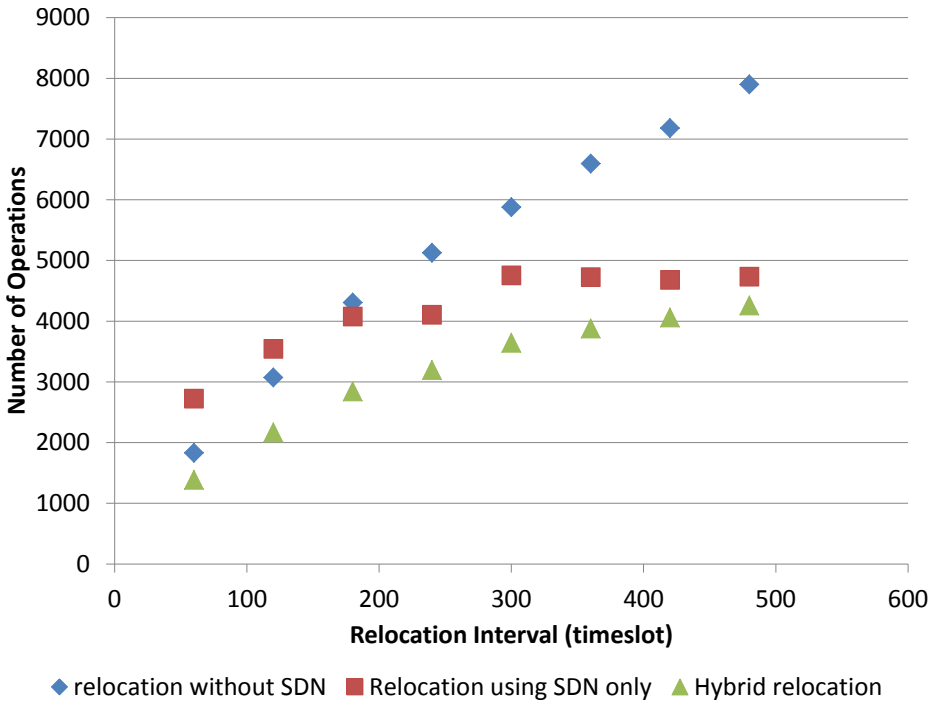

Figure 37: Comparison of the relocation method. [2]

results are biased by the small size of the network used [2]. In order to correct it for real networks, a coefficient adapted to the network situation, depending on the SDN controller load for example, can be set in order to promote a method over the other. The third method then provides a way to take advantage of both methods even if the load on the controller is already significant.

Considering that the cost of an operation on a switch is similar to the cost of an operation on the analyzer, both being reconfigured over the network by a controller, Figure 38 presents the proportion of relocations processed with the SDN method and the non-SDN method. As expected of the previous results, the use of the non-SDN method is much higher with short relocation periods and much lower with high relocation periods than the use of the SDN method [2].

Thus, Figure 39 presents the average number of operations for each method, and separated for the third one, the operations on the analyzers (proxies) and on the switches. The hybrid average number of operations per relocation is lower than the minimum average number of operations per relocation of the two methods separately, since for any cases when it would be higher for the non-SDN method, the relocation is performed using SDN and vice versa [2].

5.4.5 Forewarn

Those results are theoretical, they are based on a simulation. They must now be validated through experiments with the testbed that is currently under development.

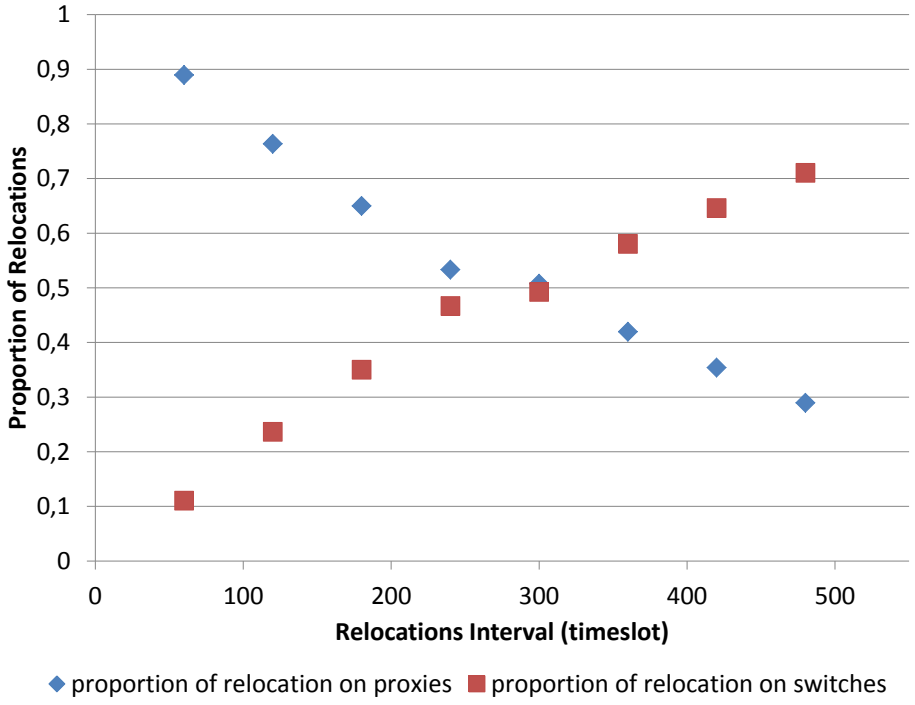

Figure 38: Proportion of each method in the relocation processes. [2]

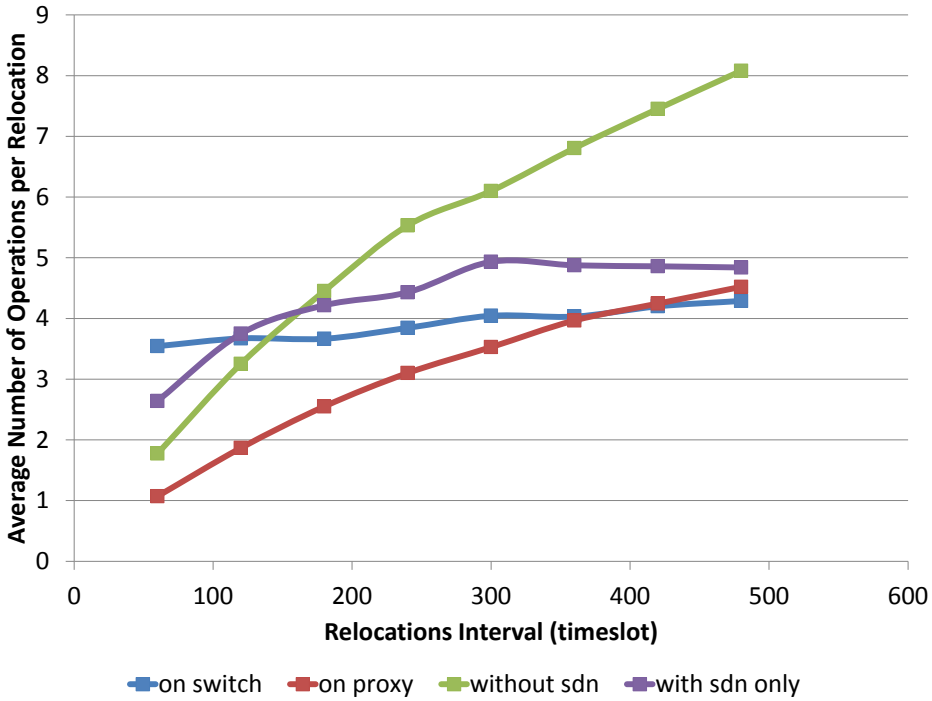

Figure 39: Average Number of Operation per relocation. [2]

5.5 Miscellaneous optimizations

Due to the flaws of the first implementation, mostly with regard to performances, another solution has been designed. This one is based on the hybrid solution, which is using IP addresses to identify the content and forward it to the correct cache while using both capabilities to perform the relocation of the content. The cache and the proxy elements have been merged into a single instance, using Apache TrafficServer as the cache engine. The cache controller is composed of two modules, a responder replying to the cache requests, built as a FastCGI application, and a relocation manager built as a daemon. Both are relying on a PostgreSQL database. [2]

This implementation includes some CDN-related features. TrafficServer has the capability to reserve dedicated storage for a domain. Thus it is possible to dedicate a storage for providers such as Akamai. The proposed system permits to bring the dedicated storage of the CDN providers closer to the end user while keeping a similar hit ratio, at equal storage capabilities, by dispatching the contents in the different locations to best fit the demand. So this system natively includes features for CDN support.

5.6 Charging for the cached content

Due to the location of the caches, the traffic from the users going to the caches does not go through the P-GW any longer. The users' consumption was previously quantified in the P-GW and thus must be calculated in another way. Thus, we propose a feature for measuring the users' consumption. When a user is requesting a piece of content that is stored locally in a cache, the size of the document fetched is added to his consumption on the cache. We only take into account what is effectively downloaded. If the object is not fully downloaded, only the downloaded part will be taken into account for the charging. Then the cache controller gathers the consumption of the clients on all the switch and stores it into its database. The PCRF system can then retrieve the consumption of every users through a dedicated interface.

5.7 Summary

We proposed three different designs to optimize the caching system. The first design is using SDN to forward the requests based on the name of their objects. This solution is making full use of the SDN capabilities, since the network takes in charge the forwarding of the requests based on their identifiers. This solution is not the most efficient in terms of number of operations. It also implies a high number of rules on the switches. The second design is mainly based on the proxies, updating their mapping tables for the relocations. It is performing better than the previous solution in terms of number of operations. The load on the SDN controller is also low. Thus we tried to improve this solution by adding the use of SDN when it could improve the performances. We proposed an hybrid solution. We studied the theoretical performances of both of them to find out that the hybrid is the most efficient design. Our optimizations permit to increase the network load reduction

and are highly adaptable to the real network situations. Those results must still be validated by experiments.

6 Conclusion

Mobile operators are currently interested into deploying cache solutions to bring the contents closer to the users. The current technology employed in LTE networks allows caching either in base stations or after the P-GW, that are both endpoints of the GTP tunnels. Caching in base stations is not performing so well because of the small number of users per base stations and their mobility. Thus the cache-hit rate is lower in the base station than in any other place of the network. In the meantime, caching after the P-GW is performing well in terms of cache-hits, but it is further away from the clients and thus leads to latency for the users and network load for the operators. Using Software-Defined Networking permits to remove the tunneling and perform caching in backhaul networks. Thus it offers a compromise between both situations. [2]

SDN can also be used to improve the caching systems themselves. Several proposals have been designed to reach this goal. The current trend in caching is to decentralize the caches and initiate a cooperation between the cache neighbors. SDN can participate in that goal, by providing a layer performing the routing or forwarding of the requests to the right caches. This can be performed in different ways, using a new protocol, extending the IP protocol or building an overlay on IP networks.

Caching in backhaul networks

Therefore we propose a new system to provide in-network caching in the backhaul networks, using the capabilities of Software-Defined Networking. We firstly present a solution using a redirection based on the IP. We implemented a testbed to validate our first prototype. It is performing as expected, reducing the load on the backhaul network, but presents some flaws in the design. We analyzed its performances and, based on the results, we designed an optimized solution which has been validated through simulations.

The use of SDN allows us to dynamically relocate the contents from one cache to another located in any part of the LTE network. The design we propose is based on a forwarding-on-content solution. It attributes an IP identifier to the objects and forwards the requests based on that identifier. Our proposed solution permits to aggregate the rules on the switches and to perform redirection with an object-level granularity. It also includes a feature in the proxies to accept updates from the controller. Our relocation method uses either the feature of the proxies or the SDN capabilities to process the change, depending on their efficiency. This solution performs better than any of the two methods separately.

Our design permits to place the content at optimal location, minimizing the bandwidth consumption and thus the costs. Then the relocation method proposed permits to keep the contents at their optimal location with the lowest costs. This improves the quality of experience (QoE) of the users and save power since the content is downloaded faster. Those benefits will be further studied.

Moreover, the system is not dependent of a single location decision algorithm.

It can use any algorithm implemented for that purpose. This is aimed at offering new caching possibilities for mobile operators in order to reduce the resource consumptions the most possible. The obvious solution would be to place the cache closer to the user, but the results show that based on the number of users, a more centralized location of the cache can reduce more the overall network load if it is selected accurately.

Another important remark is the relevance of SDN to deploy the proposed solution. The role of SDN is primarily to enable the possibility of removing the GTP tunneling, thus removing all limitations to place the cache in any part of the network. Besides this feature, SDN should not have any major role in caching but previous work and our results show that using SDN can still reduce the number of operations required to reallocate the caches. A final result worth considering is the intervals between cache reallocation which seems to be dependent on the content. This is left for further work since it requires further analysis in order to identify the optimal interval between cache reallocation.

Finally, this work suffers from a major limitation of transparent caching, it does not support HTTPS. An increasing part of the websites is using HTTPS. 25.1% of the 153257 most popular websites were using HTTPS on July 3^{rd}, 2014 [32]. Due to the security mechanisms of this protocol, it is no longer possible to intercept transparently the traffic and serve a cached version. TrafficServer supports https for the clients if they are configured to use a TrafficServer instance as SSL proxy [33], but it is not possible to use a transparent proxy in HTTPS since it would be considered as a Man-In-The-Middle attack.

Future works

This thesis opens a wide range of possible future work. The first point for future work is the implementation and testing of the current optimized design. We are willing to validate our design by experiments in a real LTE network. The second point on which we are willing to work is to extend our system, by proposing new location selection or relocation triggering algorithms. Our current location selection algorithm is exclusively based on the previous requests. So our system could be improved by proposing new location selection algorithm, for example based on a prediction of the requests. Techniques of this kind have already been implemented in replacement algorithms in caches, that are not only web caches. It could be of high interest to adapt such algorithm to select the cache location in our system . A similar point for future work would be to improve the relocation triggering algorithm. Currently, the relocation triggering algorithm is run based on time intervals. The optimal time interval seems to be dependent on the type of the content such as video, text, application; it is also dependent on the frequency of the requests per object. Thus future work could study those parameters and improve the relocation algorithm.

Wider future work could include an integration with a work on efficiency in mobile networks. The project is focusing on making the web page downloads efficient by bundling the contents, compressing the headers and selectively compress the

content [34]. Since we are using proxies to analyze the requests of the users, it makes it a perfect place to proceed to such operations. Thus we could integrate this solution in our design to further improve the overall efficiency and the users' Quality of Experience.

References

[1] M. Hibberd, "Tellabs "death clock" predicts end of profit for mnos," Feb. 2011. [Online]. Available: http://www.telecoms.com/24392/tellabs-death-clock-predicts-end-of-profit-for-mnos/

[2] M. Kimmerlin, J. Costa-Requena, and J. Manner, "Caching using software-defined networking in lte networks," in *IEEE International Conference on Advanced Networks and Telecommunications Systems*, Dec. 2014, submitted.

[3] S. H. Kim, "Intelligent caching solution to address quality-of-experience and data deluge in mobile multimedia delivery networks," 2013. [Online]. Available: http://www.lsi.com/downloads/Public/Communication%20Processors/Axxia%20Communication%20Processor/LSI_WP_IntelligentCachingMobileNetworks_v2.pdf

[4] J. Costa-Requena, "Sdn integration in lte mobile backhaul networks," in *Information Networking (ICOIN), 2014 International Conference on*, Feb. 2014, pp. 264–269.

[5] A. Basta, W. Kellerer, M. Hoffmann, K. Hoffmann, and E.-D. Schmidt, "A virtual sdn-enabled lte epc architecture: A case study for s-/p-gateways functions," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, Nov. 2013, pp. 1–7.

[6] L. Li, Z. Mao, and J. Rexford, "Toward software-defined cellular networks," in *Software Defined Networking (EWSDN), 2012 European Workshop on*, Oct. 2012, pp. 7–12.

[7] Y. Mao, Z. Zhu, and W. Shi, "Peer-to-peer web caching: hype or reality?" in *Parallel and Distributed Systems, 2004. ICPADS 2004. Proceedings. Tenth International Conference on*, Jul. 2004, pp. 171–178.

[8] A. Buhmann and J. Klein, "Examining the performance of a constraint-based database cache," in *Database Engineering and Applications Symposium, 2007. IDEAS 2007. 11th International*, Sep. 2007, pp. 290–295.

[9] J. Wang, "A survey of web caching schemes for the internet," *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 5, pp. 36–46, Oct. 1999. [Online]. Available: http://doi.acm.org/10.1145/505696.505701

[10] B. Ramanan, L. Drabeck, M. Haner, N. Nithi, T. Klein, and C. Sawkar, "Cacheability analysis of http traffic in an operational lte network," in *Wireless Telecommunications Symposium (WTS), 2013*, Apr. 2013, pp. 1–8.

[11] X. Cai, S. Zhang, and Y. Zhang, "Economic analysis of cache location in mobile network," in *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*, Apr. 2013, pp. 1243–1248.

[12] S. Woo, E. Jeong, S. Park, J. Lee, S. Ihm, and K. Park, "Comparison of caching strategies in modern cellular backhaul networks," in *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '13. ACM, 2013, pp. 319–332. [Online]. Available: http://doi.acm.org/10.1145/2462456.2464442

[13] K. Y. Lai, Z. Tari, and P. Bertok, "Supporting user mobility through cache relocation," *Mob. Inf. Syst.*, vol. 1, no. 4, pp. 275–307, Dec. 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1233697.1233700

[14] A. Arvidsson, A. MihÃ¡ly, and L. Westberg, "Optimised local caching in cellular mobile networks," *Computer Networks*, vol. 55, no. 18, pp. 4101 – 4111, 2011, internet-based Content Delivery. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128611002775

[15] J. Z. Wang, Z. Du, and P. K. Srimani, "Cooperative proxy caching for wireless base stations," *Mob. Inf. Syst.*, vol. 3, no. 1, pp. 1–18, Jan. 2007. [Online]. Available: http://dl.acm.org/citation.cfm?id=1376597.1376598

[16] J. Wang and V. Bhulawala, "A p2p cooperative proxy cache system for wireless base stations," in *Wireless Networks, Communications and Mobile Computing, 2005 International Conference on*, vol. 2, Jun. 2005, pp. 1148–1153 vol.2.

[17] N. Melazzi, A. Detti, G. Mazza, G. Morabito, S. Salsano, and L. Veltri, "An openflow-based testbed for information centric networking," in *Future Network Mobile Summit (FutureNetw), 2012*, Jul. 2012, pp. 1–9.

[18] G. Rossini and D. Rossi, "A dive into the caching performance of content centric networking," in *Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2012 IEEE 17th International Workshop on*, Sep. 2012, pp. 105–109.

[19] OpenvSwitch Community, "Openvswitch," 2014. [Online]. Available: http://openvswitch.org/

[20] M. Vahlenkamp, F. Schneider, D. Kutscher, and J. Seedorf, "Enabling information centric networking in ip networks using sdn," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, Nov. 2013, pp. 1–6.

[21] X. Nguyen, D. Saucez, and T. Turletti, "Efficient caching in content-centric networks using openflow," in *INFOCOM, 2013 Proceedings IEEE*, Apr. 2013, pp. 1–2.

[22] X. N. Nguyen, D. Saucez, and T. Turletti, "Providing ccn functionalities over openflow switches," Research report, Aug. 2013. [Online]. Available: http://hal.inria.fr/hal-00920554

[23] A. "Chanda and C. Westphal, ""contentflow: Mapping content to flows in software defined networks"," *"arXiv preprint arXiv:1302.1493"*, 2013.

[24] Y. Sakurauchi, R. McGeer, and H. Takada, "Open web: Seamless proxy interconnection at the switching layer," in *Networking and Computing (ICNC), 2010 First International Conference on*, Nov. 2010, pp. 285–289.

[25] J. Costa-Requena, M. Kimmerlin, and J. Manner, "Sdn optimized caching in lte mobile networks," in *International Conference on ICT Convergence 2014*, Oct. 2014, submitted and accepted.

[26] S. Souders, "Http interesting stats," 2014. [Online]. Available: http://httparchive.org/interesting.php

[27] MozillaZine, "Network.http.max-connections," 2011. [Online]. Available: http://kb.mozillazine.org/Network.http.max-connections

[28] T. White and al., "ionrock/cachecontrol. github," 2014. [Online]. Available: https://github.com/ionrock/cachecontrol

[29] K. Reitz, "Requests: Http for humans," 2014. [Online]. Available: http://docs.python-requests.org/en/latest/

[30] Apple, "Http live streaming overview," 2014. [Online]. Available: https://developer.apple.com/library/ios/documentation/networkinginternet/conceptual/streamingmediaguide/Introduction/Introduction.html

[31] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: evidence and implications," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, Mar. 1999, pp. 126–134 vol.1.

[32] T. trustworthy Internet Movement, "Ssl pulse july 03, 2014," 2014. [Online]. Available: https://www.trustworthyinternet.org/ssl-pulse/

[33] Apache Foundation, "Trafficserver, security options," 2013. [Online]. Available: https://docs.trafficserver.apache.org/en/latest/admin/security-options.en.html#client-and-traffic-server-connections

[34] L. Wang and J. Manner, "Energy-efficient mobile web in a bundle," *Computer Networks*, vol. 57, no. 17, pp. 3581 – 3600, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128613002624